

1.0 Data Management and Governance	3
1.1 Standard Statistical Distributions	4
1.1.1 Types of Distributions	12
1.2 Normal Distribution	18
1.3 Data Quality Issues Through Plots	54
1.4 Resolution of Data Quality Issues	61
1.5 Data Prep and Transformations	67
1.6 Identifying Data Transformation Functions Part 1	94
1.7 Identifying Data Transformation Functions Part 2	103
1.8 Normalization	113
1.9 CASE Expressions	128
1.9.1 CASE Expressions Examples	156
1.10 Connecting to External Sources	176
1.11 Creating High Performance Tables	182
2.0 Data Visualization & Presentation	227
2.1 SQL Configuration and Performance Optimizations	228
2.2 Data Visualizations	262
2.2.1 Custom Visualizations in AppCenter	282
2.3.1 Misleading Graphs	303
2.3.2 Boxplot Caveats	304
2.3.3 Calculation Errors	308
2.3.4 Histogram Issues	314
2.3.5 Dual Axis Issues	317
2.4 AppCenter Visualization Formats and Types	328
3.0 Statistical Techniques	330
3.1.1 Scatterplots and Correlation	331
3.1.2 Histogram Issues	345
3.1.3 Assumption Linearity	348
3.2 Statistical Analysis for Univariate Statistics	350
3.3 Hypothesis Testing	362

3.4 GLM Stats Model	368
3.4.1 GLM Stats Model Outcome and Significance	372
3.5 Linear Regression	412
3.5.1 Simple Linear Regression Model	429
4.0 Data Analytics Methods & Algorithms	433
4.1 Text Analysis Function Reference	434
4.2 Text Analytics Function Reference ML Engine	536
4.3 Sentiment Extractor	604
4.4 Named Entity Recognition	640
4.5 nPath	664
4.5.1 nPath Advanced	714
4.6 nPath Function Reference	733
4.7 Sessionize	768
4.8 Time Series	799
4.8.1 Time Series Detail	812
4.9 Time Series	825
4.9.1 Time Series Aggregation	838
4.10 Windowing Functions	933
4.11 CFilter	1033
4.12 CFilter Syntax and Examples	1084
5.0 Validation and Evaluation	1093
5.1 ROC Introduction	1094
5.1.1 ROC Function Reference	1098
5.2 ROC Overview	1127
5.2.1 ROC Curve and AUC Classification	1138
5.3 Classifier Thresholds	1141

# 1.0 Data Management and Governance

# Standard Statistical Distributions (e.g. Normal, Poisson, Binomial) and their uses

**Source:** <https://www.healthknowledge.org.uk/public-health-textbook/research-methods/1b-statistical-methods/statistical-distributions>

## ***Statistics: Distributions***

### **Summary**

Normal distribution describes continuous data which have a symmetric distribution, with a characteristic 'bell' shape.

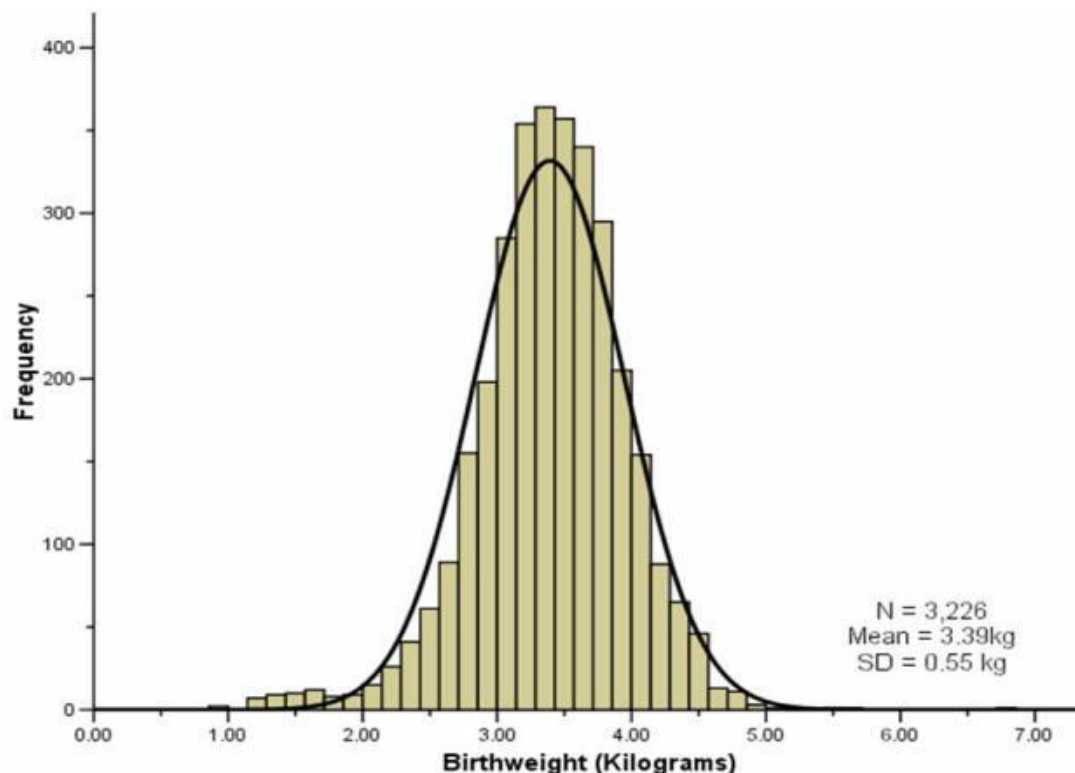
Binomial distribution describes the distribution of binary data from a finite sample. Thus it gives the probability of getting  $r$  events out of  $n$  trials.

Poisson distribution describes the distribution of binary data from an infinite sample. Thus it gives the probability of getting  $r$  events in a population.

### **The Normal Distribution**

It is often the case with medical data that the histogram of a continuous variable obtained from a single measurement on different subjects will have a characteristic 'bell-shaped' distribution known as a Normal distribution. One such example is the histogram of the birth weight (in kilograms) of the 3,226 new born babies shown in Figure 1.

*Figure 1 Distribution of birth weight in 3,226 newborn babies (data from O' Cathain et al 2002)*



To distinguish the use of the same word in normal range and Normal distribution we have used a lower and upper case convention throughout.

The histogram of the sample data is an estimate of the population distribution of birth weights in new born babies. This population distribution can be estimated by the superimposed smooth 'bell-shaped' curve or 'Normal' distribution shown. We presume that if we were able to look at the entire population of new born babies then the distribution of birth weight would have exactly the Normal shape. We often infer, from a sample whose histogram has the approximate Normal shape, that the population will have exactly, or as near as makes no practical difference, that Normal shape.

The Normal distribution is completely described by two parameters  $\mu$  and  $\sigma$ , where  $\mu$  represents the population mean, or centre of the distribution, and  $\sigma$  the population standard deviation. It is symmetrically distributed around the mean. Populations with small values of the standard deviation  $\sigma$  have a distribution concentrated close to the centre  $\mu$ ; those with large standard deviation have a distribution widely spread along the measurement axis. One mathematical property of the Normal distribution is that exactly 95% of the distribution lies between

$$\mu - (1.96 \times \sigma) \text{ and } \mu + (1.96 \times \sigma)$$

Changing the multiplier 1.96 to 2.58, exactly 99% of the Normal distribution lies in the corresponding interval.

In practice the two parameters of the Normal distribution,  $\mu$  and  $\sigma$ , must be estimated from the sample data. For this purpose a random sample from the population is first taken. The sample mean  $\bar{x}$  and the sample standard deviation,  $SD(\bar{x}) = \text{SSD}(\bar{x}) = S$ , are then calculated. If a sample is taken from such a

Normal distribution, and provided the sample is not too small, then approximately 95% of the sample lie within the interval:

$$\bar{x} - [1.96 \times SD(\bar{x})] \text{ to } \bar{x} + [1.96 \times SD(\bar{x})]$$

This is calculated by merely replacing the population parameters  $\mu$  and  $\sigma$  by the sample estimates  $\bar{x}$  and  $s$  in the previous expression.

In appropriate circumstances this interval may estimate the reference interval for a particular laboratory test which is then used for diagnostic purposes.

We can use the fact that our sample birth weight data appear Normally distributed to calculate a reference range. We have already mentioned that about 95% of the observations (from a Normal distribution) lie within  $\pm 1.96$  SDs of the mean. So a reference range for our sample of babies, using the values given in the histogram above, is:

$$3.39 - [1.96 \times 0.55] \text{ to } 3.39 + [1.96 \times 0.55]$$

2.31kg to 4.47kg

A baby's weight at birth is strongly associated with mortality risk during the first year and, to a lesser degree, with developmental problems in childhood and the risk of various diseases in adulthood. If the data are not Normally distributed then we can base the normal reference range on the observed percentiles of the sample, i.e. 95% of the observed data lie between the 2.5 and 97.5 percentiles. In this example, the percentile-based reference range for our sample was calculated as 2.19kg to 4.43kg.

Most reference ranges are based on samples larger than 3500 people. Over many years, and millions of births, the WHO has come up with a normal birth weight range for new born babies. These ranges represent results that are acceptable in newborn babies and actually cover the middle 80% of the population distribution, i.e. the 10th to 90th centiles. Low birth weight babies are usually defined (by the WHO) as weighing less than 2500g (the 10th centile) regardless of gestational age, and large birth weight babies are defined as weighing above 4000g (the 90th centile). Hence the normal birth weight range is around 2.5kg to 4kg. For our sample data, the 10th to 90th centile range was similar, 2.75 to 4.03kg.

## The Binomial Distribution

If a group of patients is given a new drug for the relief of a particular condition, then the proportion  $p$  being successfully treated can be regarded as estimating the population treatment success rate  $\pi$ .

The sample proportion  $p$  is analogous to the sample mean  $\bar{x}$ , in that if we score zero for those  $s$  patients who fail on treatment, and 1 for those  $r$  who succeed, then  $p = r/n$ , where  $n = r + s$  is the total number of patients treated. Thus  $p$  also represents a mean.

Data which can take only a binary (0 or 1) response, such as treatment failure or treatment success, follow the binomial distribution provided the underlying population response rate does not change. The binomial probabilities are calculated from:

$$P(r \text{ responses out of } n) = \frac{n!}{r!(n-r)!} \pi^r (1-\pi)^{n-r}$$

...for successive values of  $R$  from 0 through to  $n$ . In the above,  $n!$  is read as “ $n$  factorial” and  $r!$  as “ $r$  factorial”. For  $r=4$ ,  $r!=4 \times 3 \times 2 \times 1=24$ . Both  $0!$  and  $1!$  are taken as equal to 1. The shaded area marked in Figure 2 (below) corresponds to the above expression for the binomial distribution calculated for each of  $r=8,9,\dots,20$  and then added. This area totals 0.1018. So the probability of eight or more responses out of 20 is 0.1018.

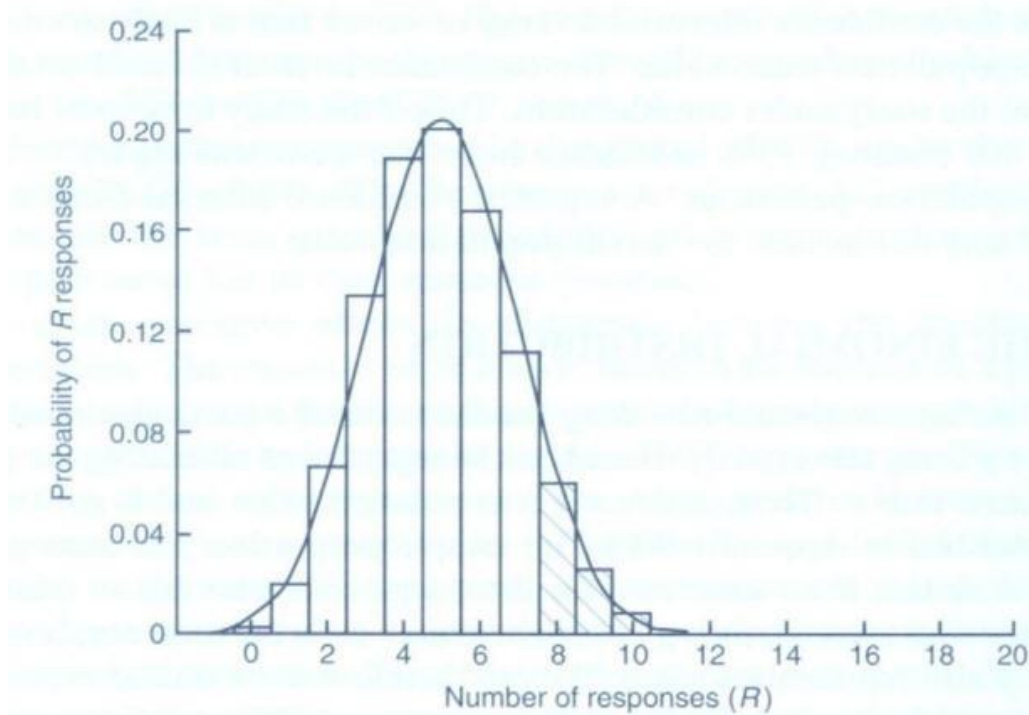
For a fixed sample size  $n$  the shape of the binomial distribution depends only on  $\pi$ . Suppose  $n = 20$  patients are to be treated, and it is known that on average a quarter, or  $\pi=0.25$ , will respond to this particular treatment. The number of responses actually observed can only take integer values between 0 (no responses) and 20 (all respond). The binomial distribution for this case is illustrated in Figure 2. The distribution is not symmetric, it has a maximum at five responses and the height of the blocks corresponds to the probability of obtaining the particular number of responses from the 20 patients yet to be treated. It should be noted that the expected value for  $r$ , the number of successes yet to be observed if we treated  $n$  patients, is  $n\pi$ . The potential variation about this expectation is expressed by the corresponding standard deviation:

$$SD(r)=n\pi(1-\pi) \quad \text{-----} \quad \sqrt{SD(r)}=\sqrt{n\pi(1-\pi)}$$

Figure 2 also shows the Normal distribution arranged to have  $\mu = n\pi = 5$  and  $\sigma = \sqrt{n\pi(1-\pi)} = 1.94$ , superimposed on to a binomial distribution with  $\pi = 0.25$  and  $n = 20$ . The Normal distribution describes fairly precisely the binomial distribution in this case. If  $n$  is small, however, or  $\pi$  close to 0 or 1, the disparity between the Normal and binomial distributions with the same mean and standard deviation increases and the Normal distribution can no longer be used to approximate the binomial distribution. In such cases the probabilities generated by the binomial distribution itself must be used.

It is also only in situations in which reasonable agreement exists between the distributions that we would use the confidence interval expression given previously. For technical reasons, the expression given for a confidence interval for a proportion is an approximation. The approximation will usually be quite good provided  $p$  is not too close to 0 or 1, situations in which either almost none or nearly all of the patients respond to treatment. The approximation improves with increasing sample size  $n$ .

*Figure 2: Binomial distribution for  $n=20$  with  $\pi=0.25$  and the Normal approximation*



### The Poisson Distribution

The Poisson distribution is used to describe discrete quantitative data such as counts in which the population size  $n$  is large, the probability of an individual event  $\pi$  is small, but the expected number of events,  $n\pi$ , is moderate (say five or more). Typical examples are the number of deaths in a town from a particular disease per day, or the number of admissions to a particular hospital.

#### Example

Wight et al (2004) looked at the variation in cadaveric heart beating organ donor rates in the UK. They found that there were 1330 organ donors, aged 15-69, across the UK for the two years 1999 and 2000 combined. Heart-beating donors are patients who are seriously ill in an intensive care unit (ICU) and are placed on a ventilator.

Now it is clear that the distribution of the number of donors takes integer values only, thus the distribution is similar in this respect to the binomial. However, there is no theoretical limit to the number of organ donors that could happen on a particular day. Here the population is the UK population aged 15-69, over two years, which is over 82 million person years, so in this case each member can be thought to have a very small probability of actually suffering an event, in this case being admitted to a hospital ICU and placed on a ventilator with a life threatening condition.

The mean number of organ donors per day over the two year period is calculated as:

$$r = 1330(365 + 365) = 1330730 = 1.82r = 1330(365 + 365) = 1330730 = 1.82 \text{ organ donations per day}$$

It should be noted that the expression for the mean is similar to that for  $\pi$ , except here multiple data values are common; and so instead of writing each as a distinct figure in the numerator they are first grouped and counted. For data arising from a Poisson distribution the standard error, that is the

standard deviation of  $r$ , is estimated by  $SE(r) = \sqrt{r/n}$ , where  $n$  is the total number of days (or an alternative time unit). Provided the organ donation rate is not too low, a 95% confidence interval for the underlying (true) organ donation rate  $\lambda$  can be calculated in the usual way:

$$r - [1.96 \times SE(r)] \text{ to } r + [1.96 \times SE(r)]$$

In the above example  $r=1.82$ ,  $SE(r)=\sqrt{1.82/730}=0.05$ , and therefore the 95% confidence interval for  $\lambda$  is 1.72 to 1.92 organ donations per day. Exact confidence intervals can be calculated as described by Altman et al. (2000).

The Poisson probabilities are calculated from:

$$P(r \text{ responses}) = \frac{\lambda^r}{r!} e^{-\lambda}$$

...for successive values of  $r$  from 0 to infinity. Here  $e$  is the exponential constant 2.7182..., and  $\lambda$  is the population rate which is estimated by  $r$  in the example above.

### Example

Suppose that before the study of Wight et al. (2004) was conducted it was expected that the number of organ donations per day was approximately two. Then assuming  $\lambda = 2$ , we would anticipate the probability of 0 organ donations in a given day to be  $(2^0/0!)e^{-2} = e^{-2} = 0.135$ . (Remember that  $2^0$  and  $0!$  are both equal to 1.) The probability of one organ donation would be  $(2^1/1!)e^{-2} = 2(e^{-2}) = 0.271$ . Similarly the probability of two organ donations per day is  $(2^2/2!)e^{-2} = 2(e^{-2}) = 0.271$ ; and so on to give for three donations 0.180, four donations 0.090, five donations 0.036, six donations 0.012, etc. If the study is then to be conducted over 2 years (730 days), each of these probabilities is multiplied by 730 to give the expected number of days during which 0, 1, 2, 3, etc. donations will occur. These expectations are 98.8, 197.6, 197.6, 131.7, 26.3, 8.8 days. A comparison can then be made between what is expected and what is actually observed.

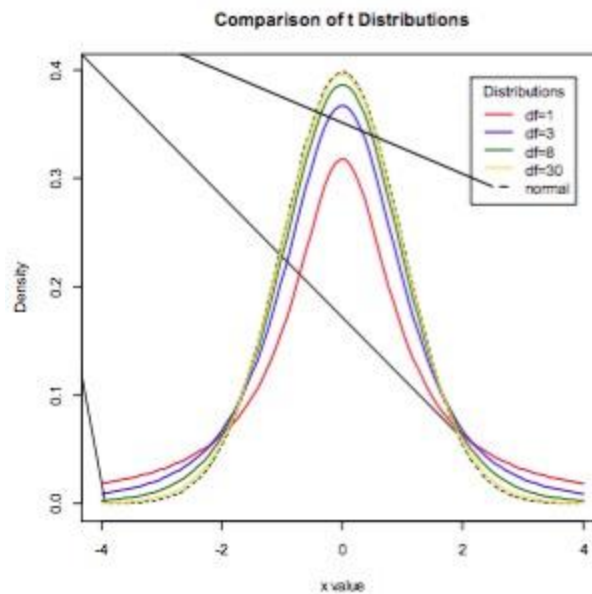
## Other Distributions

A brief description of some other distributions are given for completeness.

### t-distribution

Student's  $t$ -distribution is a continuous probability distribution with a similar shape to the Normal distribution but with wider tails.  $t$ -distributions are used to describe samples which have been drawn from a population, and the exact shape of the distribution varies with the sample size. The smaller the sample size, the more spread out the tails, and the larger the sample size, the closer the  $t$ -distribution is to the Normal distribution (Figure 3). Whilst in general the Normal distribution is used as an approximation when estimating means of samples from a Normally-distribution population, when the sample size is small (say  $n < 30$ ), the  $t$ -distribution should be used in preference.

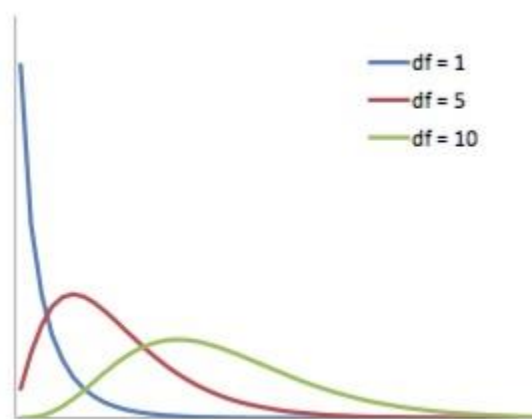
*Figure 3. The  $t$ -distribution for various sample sizes. As the sample size increases, the  $t$ -distribution more closely approximates the Normal.*



### Chi-squared distribution

The chi-squared distribution is continuous probability distribution whose shape is defined by the number of degrees of freedom. It is a right-skew distribution, but as the number of degrees of freedom increases it approximates the Normal distribution (Figure 4). The chi-squared distribution is important for its use in chi-squared tests. These are often used to test deviations between observed and expected frequencies, or to determine the independence between categorical variables. When conducting a chi-squared test, the probability values derived from chi-squared distributions can be looked up in a statistical table.

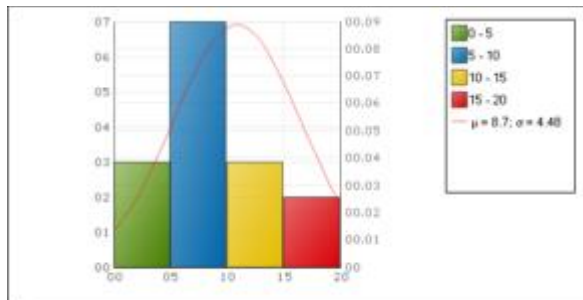
*Figure 4. The chi-squared distribution for various degrees of freedom. The distribution becomes less right-skew as the number of degrees of freedom increases.*



### Histogram

A bar diagram easy to understand but what is a histogram? Unlike a [bar graph](#) that depicts **discrete data**, histograms depict **continuous data**. The continuous data takes the form of class intervals. Thus, a histogram is a **graphical representation of a frequency distribution** with class intervals or attributes as the base and frequency as the height.

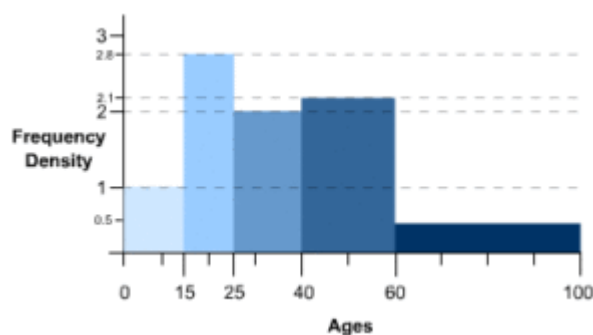
The key difference is that histograms have bars without any spaces between them and the rectangles need not be of equal width. So, we will understand histograms using an example.



In this case, see that we are considering class intervals such as 0-5, 5-10, 10-15 and 15-20. These are continuous data. In case, the class intervals given to you are not continuous, you must make it continuous first.

Here, you can interpret the histogram using the information that the graph gives. Consider the frequency to be as given on the left vertical axis and ignore the values on the right vertical axis. Thus, for the class interval 0-5, the corresponding frequency is 3. Again, for 5-10, the frequency is 7, and so on.

Note that we have taken the simple case of a histogram with bars of equal width. But as mentioned, it might not be the case if the class intervals are not even in size. In that case, you will get a histogram with bars stuck to each other (without any space between them) but with different widths. It could look something like this, but exactly how it will look depends on the [data](#):



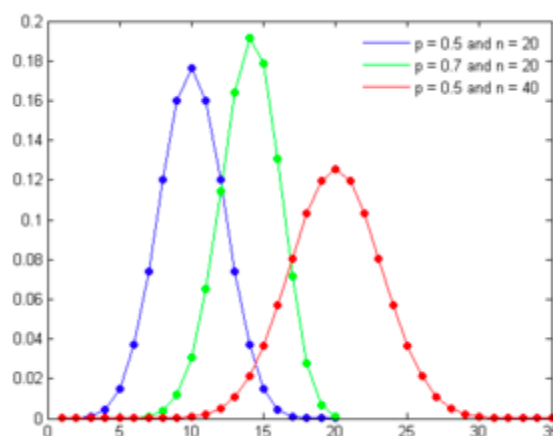
## Types of Distributions

**Source:** [http://people.stern.nyu.edu/adamodar/New\\_Home\\_Page/StatFile/statdistns.htm](http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm)

Discrete data, the entire distribution can either be developed from scratch or the data can be fitted to a pre-specified discrete distribution. With the former, there are two steps to building the distribution. The first is identifying the possible outcomes and the second is to estimate probabilities to each outcome. As we noted in the text, we can draw on historical data or experience as well as specific knowledge about the investment being analyzed to arrive at the final distribution. This process is relatively simple to accomplish when there are a few outcomes with a well-established basis for estimating probabilities but becomes more tedious as the number of outcomes increases. If it is difficult or impossible to build up a customized distribution, it may still be possible fit the data to one of the following discrete distributions:

- a. Binomial distribution: The binomial distribution measures the probabilities of the number of successes over a given number of trials with a specified probability of success in each try. In the simplest scenario of a coin toss (with a fair coin), where the probability of getting a head with each toss is 0.50 and there are a hundred trials, the binomial distribution will measure the likelihood of getting anywhere from no heads in a hundred tosses (very unlikely) to 50 heads (the most likely) to 100 heads (also very unlikely). The binomial distribution in this case will be symmetric, reflecting the even odds; as the probabilities shift from even odds, the distribution will get more skewed. Figure 6A.1 presents binomial distributions for three scenarios – two with 50% probability of success and one with a 70% probability of success and different trial sizes.

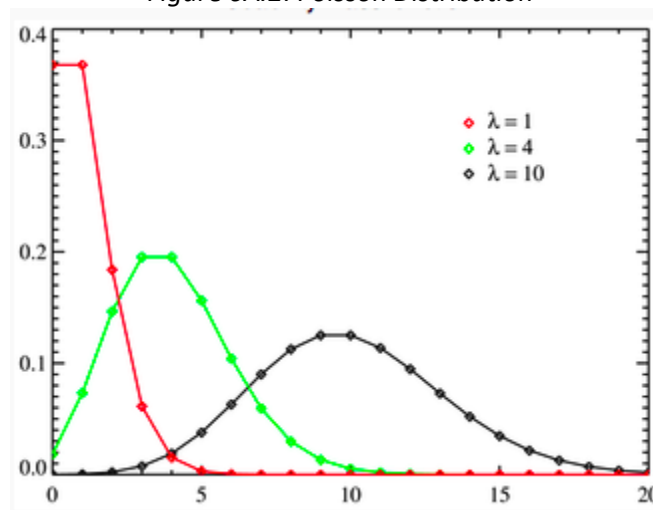
*Figure 6A.1: Binomial Distribution*



As the probability of success is varied (from 50%) the distribution will also shift its shape, becoming positively skewed for probabilities less than 50% and negatively skewed for probabilities greater than 50%.<sup>[1]</sup>

- b. Poisson distribution: The Poisson distribution measures the likelihood of a number of events occurring within a given time interval, where the key parameter that is required is the average number of events in the given interval ( $\lambda$ ). The resulting distribution looks similar to the binomial, with the skewness being positive but decreasing with  $\lambda$ . Figure 6A.2 presents three Poisson distributions, with  $\lambda$  ranging from 1 to 10.

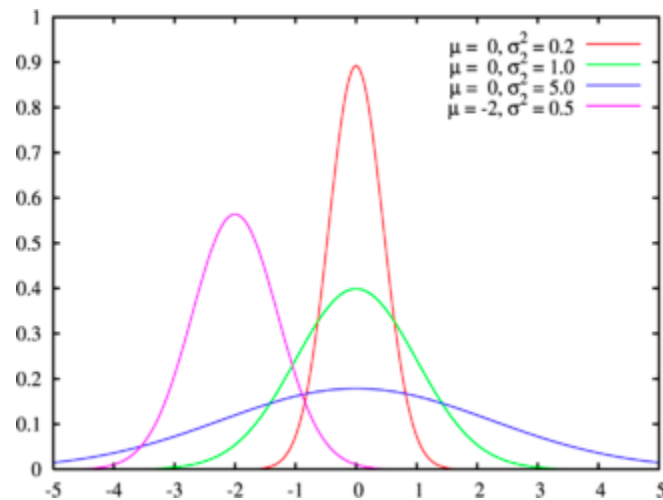
Figure 6A.2: Poisson Distribution



- c. Negative Binomial distribution: Returning again to the coin toss example, assume that you hold the number of successes fixed at a given number and estimate the number of tries you will have before you reach the specified number of successes. The resulting distribution is called the negative binomial and it very closely resembles the Poisson. In fact, the negative binomial distribution converges on the Poisson distribution, but will be more skewed to the right (positive values) than the Poisson distribution with similar parameters.

There are some datasets that exhibit symmetry, i.e., the upside is mirrored by the downside. The symmetric distribution that most practitioners have familiarity with is the normal distribution, shown in Figure 6A.6, for a range of parameters:

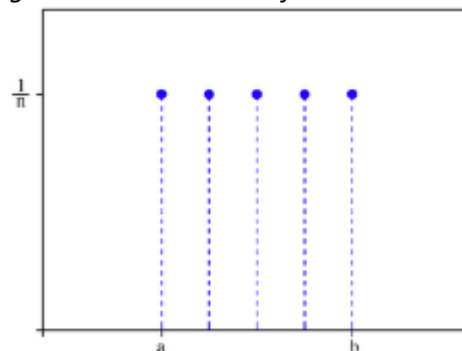
Figure 6A.6: Normal Distribution



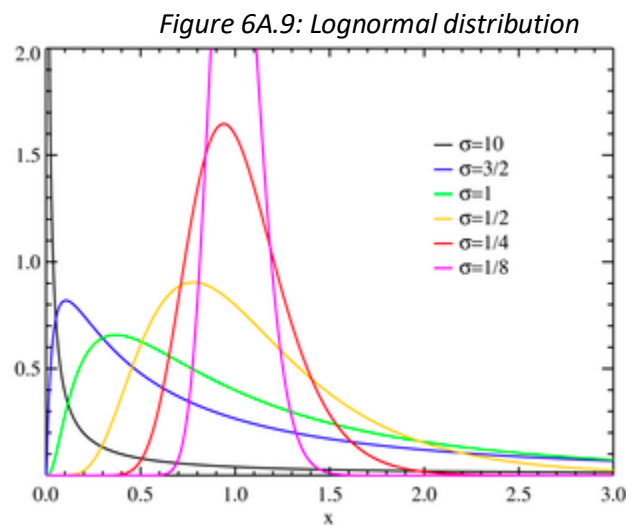
The normal distribution has several features that make it popular. First, it can be fully characterized by just two parameters – the mean and the standard deviation – and thus reduces estimation pain. Second, the probability of any value occurring can be obtained simply by knowing how many standard deviations separate the value from the mean; the probability that a value will fall 2 standard deviations from the mean is roughly 95%. The normal distribution is best suited for data that, at the minimum, meets the following conditions:

- a. There is a strong tendency for the data to take on a central value.
  - b. Positive and negative deviations from this central value are equally likely
  - c. The frequency of the deviations falls off rapidly as we move further away from the central value.
- f. Discrete uniform distribution: This is the simplest of discrete distributions and applies when all of the outcomes have an equal probability of occurring. Figure 6A.5 presents a uniform discrete distribution with five possible outcomes, each occurring 20% of the time:

Figure 6A.5: Discrete Uniform Distribution



Most data does not exhibit symmetry and instead skews towards either very large positive or very large negative values. If the data is positively skewed, one common choice is the lognormal distribution, which is typically characterized by three parameters: a shape ( $s$  or  $\sigma$ ), a scale ( $m$  or median) and a shift parameter ( $\theta$ ). When  $m=0$  and  $\theta=1$ , you have the standard lognormal distribution and when  $\theta=0$ , the distribution requires only scale and sigma parameters. As the sigma rises, the peak of the distribution shifts to the left and the skewness in the distribution increases. Figure 6A.9 graphs lognormal distributions for a range of parameters:



#### Gaussian Distribution and Reference Range

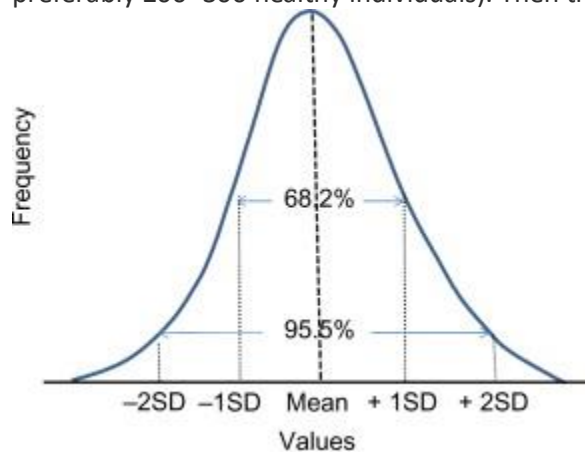
Gaussian distribution (also known as normal distribution) is a bell-shaped curve, and it is assumed that during any measurement values will follow a normal distribution with an equal number of measurements above and below the mean value. In order to understand normal distribution, it is important to know the definitions of “mean,” “median,” and “mode.” The “mean” is the calculated average of all values, the “median” is the value at the center point (mid-point) of the distribution, while the “mode” is the value that was observed most frequently during the measurement. If a distribution is normal, then the values of the mean, median, and mode are the same. However, the value of the mean, median, and mode may be different if the distribution is skewed (not Gaussian distribution). Other characteristics of Gaussian distributions are as follows:

- Mean $\pm$ 1 SD contain 68.2% of all values.

- Mean $\pm$ 2 SD contain 95.5% of all values.

Mean $\pm$ 3 SD contain 99.7% of all values.

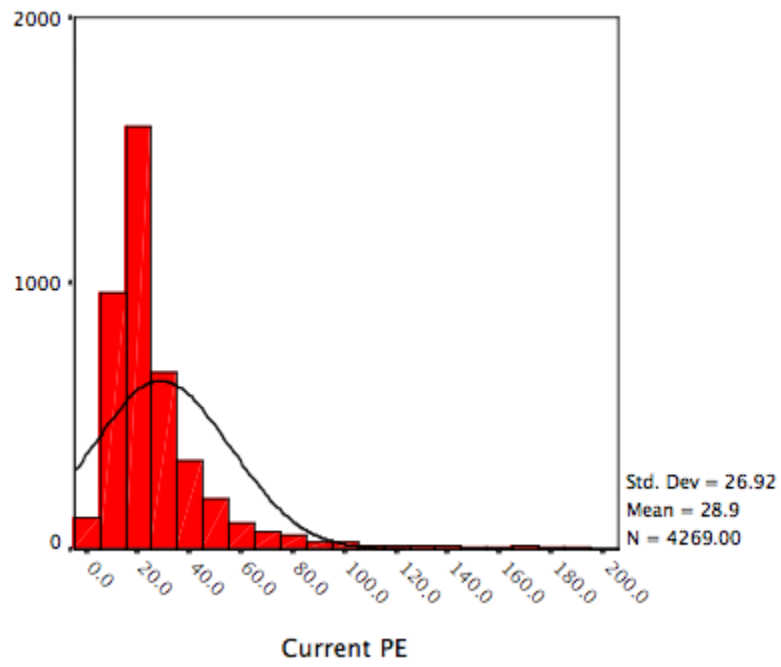
A Gaussian distribution is shown in Figure 4.1. Usually, reference range is determined by measuring the value of an analyte in a large number of normal subjects (at least 100 normal healthy people, but preferably 200–300 healthy individuals). Then the mean and standard deviations are determined.



### Tests for Fit

The simplest test for distributional fit is visual with a comparison of the histogram of the actual data to the fitted distribution. Consider figure 6A.16, where we report the distribution of current price earnings ratios for US stocks in early 2007, with a normal distribution superimposed on it.

*Figure 6A.16: Current PE Ratios for US Stocks – January 2007*



# Normal distribution

In probability theory, a **normal** (or **Gaussian** or **Gauss** or **Laplace–Gauss**) **distribution** is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

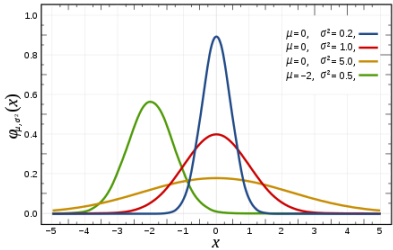
The parameter  $\mu$  is the mean or expectation of the distribution (and also its median and mode), while the parameter  $\sigma$  is its standard deviation.<sup>[1]</sup> The variance of the distribution is  $\sigma^2$ .<sup>[2]</sup> A random variable with a Gaussian distribution is said to be **normally distributed**, and is called a **normal deviate**.

Normal distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known.<sup>[3][4]</sup> Their importance is partly due to the central limit theorem. It states that, under some conditions, the average of many samples (observations) of a random variable with finite mean and variance is itself a random variable—whose distribution converges to a normal distribution as the number of samples increases. Therefore, physical quantities that are expected to be the sum of many independent processes, such as measurement errors, often have distributions that are nearly normal.<sup>[5]</sup>

Moreover, Gaussian distributions have some unique properties that are valuable in analytic studies. For instance, any linear combination of a fixed collection of normal deviates is a normal deviate. Many results and methods, such as propagation of uncertainty and least squares parameter fitting, can be derived

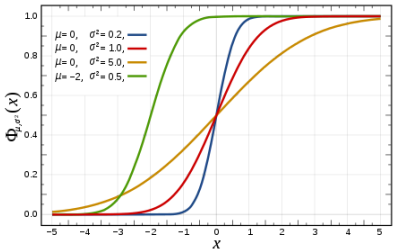
## Normal distribution

Probability density function



The red curve is the *standard normal distribution*

Cumulative distribution function



Cumulative distribution function for the normal distribution

<b>Notation</b>	$\mathcal{N}(\mu, \sigma^2)$
<b>Parameters</b>	$\mu \in \mathbb{R}$ = mean (location) $\sigma^2 > 0$ = variance (squared <u>scale</u> )
<b>Support</b>	$x \in \mathbb{R}$
<b>PDF</b>	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
<b>CDF</b>	$\frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right) \right]$
<b>Quantile</b>	$\mu + \sigma\sqrt{2} \operatorname{erf}^{-1}(2p - 1)$
<b>Mean</b>	$\mu$
<b>Median</b>	$\mu$
<b>Mode</b>	$\mu$
<b>Variance</b>	$\sigma^2$
<b>MAD</b>	$\sigma\sqrt{2/\pi}$
<b>Skewness</b>	0
<b>Ex. kurtosis</b>	0
<b>Entropy</b>	$\frac{1}{2} \log(2\pi e \sigma^2)$
<b>MGF</b>	$\exp(\mu t + \sigma^2 t^2 / 2)$
<b>CF</b>	$\exp(i\mu t - \sigma^2 t^2 / 2)$

analytically in explicit form when the relevant variables are normally distributed.

A normal distribution is sometimes informally called a **bell curve**.<sup>[6]</sup> However, many other distributions are bell-shaped (such as the Cauchy, Student's *t*, and logistic distributions).

## Contents

### Definitions

[Standard normal distribution](#)

[General normal distribution](#)

[Notation](#)

[Alternative parameterizations](#)

[Cumulative distribution function](#)

[Standard deviation and coverage](#)

[Quantile function](#)

### Properties

[Symmetries and derivatives](#)

[Moments](#)

[Fourier transform and characteristic function](#)

[Moment and cumulant generating functions](#)

[Stein operator and class](#)

[Zero-variance limit](#)

[Maximum entropy](#)

[Operations on normal deviates](#)

[Infinite divisibility and Cramér's theorem](#)

[Bernstein's theorem](#)

[Other properties](#)

### Related distributions

[Central limit theorem](#)

[Operations on a single random variable](#)

[Combination of two independent random variables](#)

[Combination of two or more independent random variables](#)

[Operations on the density function](#)

[Extensions](#)

### Statistical inference

[Estimation of parameters](#)

[Sample mean](#)

[Sample variance](#)

<b><u>Fisher information</u></b>	$\mathcal{I}(\mu, \sigma) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{pmatrix}$ $\mathcal{I}(\mu, \sigma^2) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 1/(2\sigma^4) \end{pmatrix}$
<b><u>Kullback-Leibler divergence</u></b>	$D_{\text{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2} \left\{ \left( \frac{\sigma_0}{\sigma_1} \right)^2 + \frac{(\mu_1 - \mu_0)^2}{\sigma_1^2} - 1 + 2 \ln \frac{\sigma_1}{\sigma_0} \right\}$

[Confidence intervals](#)[Normality tests](#)[Bayesian analysis of the normal distribution](#)[Sum of two quadratics](#)[Scalar form](#)[Vector form](#)[Sum of differences from the mean](#)[With known variance](#)[With known mean](#)[With unknown mean and unknown variance](#)**Occurrence and applications**[Exact normality](#)[Approximate normality](#)[Assumed normality](#)[Produced normality](#)**Computational methods**[Generating values from normal distribution](#)[Numerical approximations for the normal CDF](#)**History**[Development](#)[Naming](#)**See also****Notes****References**[Citations](#)[Sources](#)**External links**

## Definitions

---

### Standard normal distribution

The simplest case of a normal distribution is known as the *standard normal distribution*. This is a special case when  $\mu = 0$  and  $\sigma = 1$ , and it is described by this [probability density function](#):<sup>[1]</sup>

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Here, the factor  $1/\sqrt{2\pi}$  ensures that the total area under the curve  $\varphi(x)$  is equal to one.<sup>[note 1]</sup> The factor  $1/2$  in the exponent ensures that the distribution has unit variance (i.e., variance being equal to one), and therefore also unit standard deviation. This function is symmetric around  $x = 0$ , where it attains its maximum value  $1/\sqrt{2\pi}$  and has [inflection points](#) at  $x = +1$  and  $x = -1$ .

Authors differ on which normal distribution should be called the "standard" one. Carl Friedrich Gauss, for example, defined the standard normal as having a variance of  $\sigma^2 = 1/2$ . That is:

$$\varphi(x) = \frac{e^{-x^2}}{\sqrt{\pi}}$$

On the other hand, Stephen Stigler<sup>[7]</sup> goes even further, defining the standard normal as having a variance of  $\sigma^2 = 1/(2\pi)$ :

$$\varphi(x) = e^{-\pi x^2}$$

## General normal distribution

Every normal distribution is a version of the standard normal distribution, whose domain has been stretched by a factor  $\sigma$  (the standard deviation) and then translated by  $\mu$  (the mean value):

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sigma} \varphi\left(\frac{x - \mu}{\sigma}\right)$$

The probability density must be scaled by  $1/\sigma$  so that the integral is still 1.

If  $Z$  is a standard normal deviate, then  $X = \sigma Z + \mu$  will have a normal distribution with expected value  $\mu$  and standard deviation  $\sigma$ . Conversely, if  $X$  is a normal deviate with parameters  $\mu$  and  $\sigma^2$ , then the distribution  $Z = (X - \mu)/\sigma$  will have a standard normal distribution. This variate is also called the standardized form of  $X$ .

## Notation

The probability density of the standard Gaussian distribution (standard normal distribution, with zero mean and unit variance) is often denoted with the Greek letter  $\phi$  (phi).<sup>[8]</sup> The alternative form of the Greek letter phi,  $\varphi$ , is also used quite often.<sup>[1]</sup>

The normal distribution is often referred to as  $N(\mu, \sigma^2)$  or  $\mathcal{N}(\mu, \sigma^2)$ .<sup>[1][9]</sup> Thus when a random variable  $X$  is normally distributed with mean  $\mu$  and variance  $\sigma^2$ , one may write

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

## Alternative parameterizations

Some authors advocate using the precision  $\tau$  as the parameter defining the width of the distribution, instead of the deviation  $\sigma$  or the variance  $\sigma^2$ . The precision is normally defined as the reciprocal of the variance,  $1/\sigma^2$ .<sup>[10]</sup> The formula for the distribution then becomes

$$f(x) = \sqrt{\frac{\tau}{2\pi}} e^{-\tau(x-\mu)^2/2}.$$

This choice is claimed to have advantages in numerical computations when  $\sigma$  is very close to zero, and simplifies formulas in some contexts, such as in the Bayesian inference of variables with multivariate normal distribution.

Alternatively, the reciprocal of the standard deviation  $\tau' = 1/\sigma$  might be defined as the *precision*, in which case the expression of the normal distribution becomes

$$f(x) = \frac{\tau'}{\sqrt{2\pi}} e^{-(\tau')^2(x-\mu)^2/2}.$$

According to Stigler, this formulation is advantageous because of a much simpler and easier-to-remember formula, and simple approximate formulas for the quantiles of the distribution.

Normal distributions form an exponential family with natural parameters  $\theta_1 = \frac{\mu}{\sigma^2}$  and  $\theta_2 = \frac{-1}{2\sigma^2}$ , and natural statistics  $x$  and  $x^2$ . The dual expectation parameters for normal distribution are  $\eta_1 = \mu$  and  $\eta_2 = \mu^2 + \sigma^2$ .

## Cumulative distribution function

The cumulative distribution function (CDF) of the standard normal distribution, usually denoted with the capital Greek letter  $\Phi$  (phi),<sup>[1]</sup> is the integral

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

The related error function  $\text{erf}(x)$  gives the probability of a random variable, with normal distribution of mean 0 and variance 1/2 falling in the range  $[-x, x]$ . That is:<sup>[1]</sup>

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

These integrals cannot be expressed in terms of elementary functions, and are often said to be special functions. However, many numerical approximations are known; see below for more.

The two functions are closely related, namely

$$\Phi(x) = \frac{1}{2} \left[ 1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right]$$

For a generic normal distribution with density  $f$ , mean  $\mu$  and deviation  $\sigma$ , the cumulative distribution function is

$$F(x) = \Phi\left(\frac{x - \mu}{\sigma}\right) = \frac{1}{2} \left[ 1 + \text{erf}\left(\frac{x - \mu}{\sigma\sqrt{2}}\right) \right]$$

The complement of the standard normal CDF,  $Q(x) = 1 - \Phi(x)$ , is often called the Q-function, especially in engineering texts.<sup>[11][12]</sup> It gives the probability that the value of a standard normal random variable  $X$  will exceed  $x$ :  $P(X > x)$ . Other definitions of the  $Q$ -function, all of which are simple transformations of  $\Phi$ , are also used occasionally.<sup>[13]</sup>

The graph of the standard normal CDF  $\Phi$  has 2-fold rotational symmetry around the point (0,1/2); that is,  $\Phi(-x) = 1 - \Phi(x)$ . Its antiderivative (indefinite integral) can be expressed as follows:

$$\int \Phi(x) dx = x\Phi(x) + \varphi(x) + C.$$

The CDF of the standard normal distribution can be expanded by Integration by parts into a series:

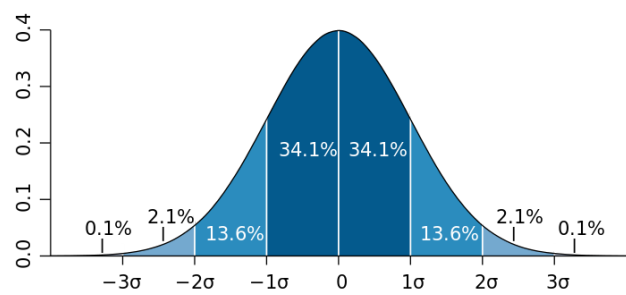
$$\Phi(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2} \left[ x + \frac{x^3}{3} + \frac{x^5}{3 \cdot 5} + \cdots + \frac{x^{2n+1}}{(2n+1)!!} + \cdots \right]$$

where **!!** denotes the double factorial.

Standard deviation and coverage

About 68% of values drawn from a normal distribution are within one standard deviation  $\sigma$  away from the mean; about 95% of the values lie within two standard deviations; and about 99.7% are within three standard deviations.<sup>[6]</sup> This fact is known as the 68-95-99.7 (empirical) rule, or the *3-sigma rule*.

More precisely, the probability that a normal deviate lies in the range between  $\mu - n\sigma$  and  $\mu + n\sigma$  is given by



For the normal distribution, the values less than one standard deviation away from the mean account for 68.27% of the set; while two standard deviations from the mean account for 95.45%; and three standard deviations account for 99.73%.

$$F(\mu + n\sigma) - F(\mu - n\sigma) = \Phi(n) - \Phi(-n) = \operatorname{erf}\left(\frac{n}{\sqrt{2}}\right).$$

To 12 significant figures, the values for  $n = 1, 2, \dots, 6$  are:<sup>[15]</sup>

$n$	$p = F(\mu + n\sigma) - F(\mu - n\sigma)$	i.e. $1 - p$	or 1 in $p$	OEIS
1	0.682 689 492 137	0.317 310 507 863	3.151 487 187 53	<a href="#">OEIS: A178647</a>
2	0.954 499 736 104	0.045 500 263 896	21.977 894 5080	<a href="#">OEIS: A110894</a>
3	0.997 300 203 937	0.002 699 796 063	370.398 347 345	<a href="#">OEIS: A270712</a>
4	0.999 936 657 516	0.000 063 342 484	15 787.192 7673	
5	0.999 999 426 697	0.000 000 573 303	1 744 277.893 62	
6	0.999 999 998 027	0.000 000 001 973	506 797 345.897	

For large  $n$ , one can use the approximation  $1 - p \approx \frac{e^{-n^2/2}}{n\sqrt{\pi/2}}$ .

Quantile function

The quantile function of a distribution is the inverse of the cumulative distribution function. The quantile function of the standard normal distribution is called the probit function, and can be expressed in terms of the inverse error function:

$$\Phi^{-1}(p) = \sqrt{2}\operatorname{erf}^{-1}(2p - 1), \quad p \in (0, 1).$$

For a normal random variable with mean  $\mu$  and variance  $\sigma^2$ , the quantile function is

$$F^{-1}(p) = \mu + \sigma\Phi^{-1}(p) = \mu + \sigma\sqrt{2}\operatorname{erf}^{-1}(2p - 1), \quad p \in (0, 1).$$

The quantile  $\Phi^{-1}(p)$  of the standard normal distribution is commonly denoted as  $z_p$ . These values are used in hypothesis testing, construction of confidence intervals and Q-Q plots. A normal random variable  $X$  will exceed  $\mu + z_p\sigma$  with probability  $1 - p$ , and will lie outside the interval  $\mu \pm z_p\sigma$  with probability  $2(1 - p)$ . In particular, the

The following table gives the quantile  $z_p$  such that  $X$  will lie in the range  $\mu \pm z_p\sigma$  with a specified probability  $p$ . These values are useful to determine tolerance interval for sample averages and other statistical estimators with normal (or asymptotically normal) distributions.<sup>[16][17]</sup> NOTE: the following table shows  $\sqrt{2}\operatorname{erf}^{-1}(p) = \Phi^{-1}\left(\frac{p+1}{2}\right)$ , not  $\Phi^{-1}(p)$  as defined above.

<i>p</i>	<i>z<sub>p</sub></i>	<i>p</i>	<i>z<sub>p</sub></i>
0.80	1.281 551 565 545	0.999	3.290 526 731 492
0.90	1.644 853 626 951	0.9999	3.890 591 886 413
0.95	1.959 963 984 540	0.99999	4.417 173 413 469
0.98	2.326 347 874 041	0.999999	4.891 638 475 699
0.99	2.575 829 303 549	0.9999999	5.326 723 886 384
0.995	2.807 033 768 344	0.99999999	5.730 728 868 236
0.998	3.090 232 306 168	0.999999999	6.109 410 204 869

For small  $p$ , the quantile function has the useful asymptotic expansion 
$$\Phi^{-1}(p) = -\sqrt{\ln \frac{1}{p^2} - \ln \ln \frac{1}{p^2} - \ln(2\pi)} + o(1).$$

## Properties

The normal distribution is the only distribution whose cumulants beyond the first two (i.e., other than the mean and variance) are zero. It is also the continuous distribution with the maximum entropy for a specified mean and variance.<sup>[18][19]</sup> Geary has shown, assuming that the mean and variance are finite, that the normal distribution is the only distribution where the mean and variance calculated from a set of independent draws are independent of each other.<sup>[20][21]</sup>

The normal distribution is a subclass of the elliptical distributions. The normal distribution is symmetric about its mean, and is non-zero over the entire real line. As such it may not be a suitable model for variables that are inherently positive or strongly skewed, such as the weight of a person or the price of a share. Such variables may be better described by other distributions, such as the log-normal distribution or the Pareto distribution.

The value of the normal distribution is practically zero when the value  $x$  lies more than a few standard deviations away from the mean (e.g., a spread of three standard deviations covers all but 0.27% of the total distribution). Therefore, it may not be an appropriate model when one expects a significant fraction of outliers—values that lie many standard deviations away from the mean—and least squares and other statistical inference methods that are optimal for normally distributed variables often become highly unreliable when applied to such data. In those cases, a more heavy-tailed distribution should be assumed and the appropriate robust statistical inference methods applied.

The Gaussian distribution belongs to the family of stable distributions which are the attractors of sums of independent, identically distributed distributions whether or not the mean or variance is finite. Except for the Gaussian which is a limiting case, all stable distributions have heavy tails and infinite variance. It is one of the few distributions that are stable and that have probability density functions that can be expressed analytically, the others being the Cauchy distribution and the Lévy distribution.

## Symmetries and derivatives

The normal distribution with density  $f(x)$  (mean  $\mu$  and standard deviation  $\sigma > 0$ ) has the following properties:

- It is symmetric around the point  $x = \mu$ , which is at the same time the mode, the median and the mean of the distribution.<sup>[22]</sup>
- It is unimodal: its first derivative is positive for  $x < \mu$ , negative for  $x > \mu$ , and zero only at  $x = \mu$ .
- The area under the curve and over the  $x$ -axis is unity (i.e. equal to one).
- Its first derivative is  $f'(x) = -\frac{x - \mu}{\sigma^2} f(x)$ .
- Its density has two inflection points (where the second derivative of  $f$  is zero and changes sign), located one standard deviation away from the mean, namely at  $x = \mu - \sigma$  and  $x = \mu + \sigma$ .<sup>[22]</sup>
- Its density is log-concave.<sup>[22]</sup>
- Its density is infinitely differentiable, indeed supersmooth of order 2.<sup>[23]</sup>

Furthermore, the density  $\varphi$  of the standard normal distribution (i.e.  $\mu = 0$  and  $\sigma = 1$ ) also has the following properties:

- Its first derivative is  $\varphi'(x) = -x\varphi(x)$ .
- Its second derivative is  $\varphi''(x) = (x^2 - 1)\varphi(x)$
- More generally, its  $n$ th derivative is  $\varphi^{(n)}(x) = (-1)^n \text{He}_n(x)\varphi(x)$ , where  $\text{He}_n(x)$  is the  $n$ th (probabilist) Hermite polynomial.<sup>[24]</sup>
- The probability that a normally distributed variable  $X$  with known  $\mu$  and  $\sigma$  is in a particular set, can be calculated by using the fact that the fraction  $Z = (X - \mu)/\sigma$  has a standard normal distribution.

## Moments

The plain and absolute moments of a variable  $X$  are the expected values of  $X^p$  and  $|X|^p$ , respectively. If the expected value  $\mu$  of  $X$  is zero, these parameters are called *central moments*. Usually we are interested only in moments with integer order  $p$ .

If  $X$  has a normal distribution, these moments exist and are finite for any  $p$  whose real part is greater than  $-1$ . For any non-negative integer  $p$ , the plain central moments are:<sup>[25]</sup>

$$\mathbb{E}[(X - \mu)^p] = \begin{cases} 0 & \text{if } p \text{ is odd,} \\ \sigma^p (p-1)!! & \text{if } p \text{ is even.} \end{cases}$$

Here  $n!!$  denotes the double factorial, that is, the product of all numbers from  $n$  to 1 that have the same parity as  $n$ .

The central absolute moments coincide with plain moments for all even orders, but are nonzero for odd orders. For any non-negative integer  $p$ ,

$$\begin{aligned} \mathbb{E}[|X - \mu|^p] &= \sigma^p (p-1)!! \cdot \begin{cases} \sqrt{\frac{2}{\pi}} & \text{if } p \text{ is odd} \\ 1 & \text{if } p \text{ is even} \end{cases} \\ &= \sigma^p \cdot \frac{2^{p/2} \Gamma\left(\frac{p+1}{2}\right)}{\sqrt{\pi}}. \end{aligned}$$

The last formula is valid also for any non-integer  $p > -1$ . When the mean  $\mu \neq 0$ , the plain and absolute moments can be expressed in terms of confluent hypergeometric functions  ${}_1F_1$  and  $U$ .

$$\begin{aligned} \mathbb{E}[X^p] &= \sigma^p \cdot (-i\sqrt{2})^p U\left(-\frac{p}{2}, \frac{1}{2}, -\frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2\right), \\ \mathbb{E}[|X|^p] &= \sigma^p \cdot 2^{p/2} \frac{\Gamma\left(\frac{1+p}{2}\right)}{\sqrt{\pi}} {}_1F_1\left(-\frac{p}{2}, \frac{1}{2}, -\frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2\right). \end{aligned}$$

These expressions remain valid even if  $p$  is not integer. See also generalized Hermite polynomials.

Order	Non-central moment	Central moment
1	$\mu$	0
2	$\mu^2 + \sigma^2$	$\sigma^2$
3	$\mu^3 + 3\mu\sigma^2$	0
4	$\mu^4 + 6\mu^2\sigma^2 + 3\sigma^4$	$3\sigma^4$
5	$\mu^5 + 10\mu^3\sigma^2 + 15\mu\sigma^4$	0
6	$\mu^6 + 15\mu^4\sigma^2 + 45\mu^2\sigma^4 + 15\sigma^6$	$15\sigma^6$
7	$\mu^7 + 21\mu^5\sigma^2 + 105\mu^3\sigma^4 + 105\mu\sigma^6$	0
8	$\mu^8 + 28\mu^6\sigma^2 + 210\mu^4\sigma^4 + 420\mu^2\sigma^6 + 105\sigma^8$	$105\sigma^8$

The expectation of  $X$  conditioned on the event that  $X$  lies in an interval  $[a, b]$  is given by

$$\mathbb{E}[X \mid a < X < b] = \mu - \sigma^2 \frac{f(b) - f(a)}{F(b) - F(a)}$$

where  $f$  and  $F$  respectively are the density and the cumulative distribution function of  $X$ . For  $b = \infty$  this is known as the inverse Mills ratio. Note that above, density  $f$  of  $X$  is used instead of standard normal density as in inverse Mills ratio, so here we have  $\sigma^2$  instead of  $\sigma$ .

## Fourier transform and characteristic function

The Fourier transform of a normal density  $f$  with mean  $\mu$  and standard deviation  $\sigma$  is<sup>[26]</sup>

$$\hat{f}(t) = \int_{-\infty}^{\infty} f(x) e^{-itx} dx = e^{-i\mu t} e^{-\frac{1}{2}(\sigma t)^2}$$

where  $i$  is the imaginary unit. If the mean  $\mu = 0$ , the first factor is 1, and the Fourier transform is, apart from a constant factor, a normal density on the frequency domain, with mean 0 and standard deviation  $1/\sigma$ . In particular, the standard normal distribution  $\varphi$  is an eigenfunction of the Fourier transform.

In probability theory, the Fourier transform of the probability distribution of a real-valued random variable  $X$  is closely connected to the characteristic function  $\varphi_X(t)$  of that variable, which is defined as the expected value of  $e^{itX}$ , as a function of the real variable  $t$  (the frequency parameter of the Fourier transform). This definition can be analytically extended to a complex-value variable  $t$ .<sup>[27]</sup> The relation between both is:

$$\varphi_X(t) = \hat{f}(-t)$$

## Moment and cumulant generating functions

The moment generating function of a real random variable  $\mathbf{X}$  is the expected value of  $e^{t\mathbf{X}}$ , as a function of the real parameter  $t$ . For a normal distribution with density  $f$ , mean  $\mu$  and deviation  $\sigma$ , the moment generating function exists and is equal to

$$M(t) = \mathbb{E}[e^{t\mathbf{X}}] = \hat{f}(it) = e^{\mu t} e^{\frac{1}{2}\sigma^2 t^2}$$

The cumulant generating function is the logarithm of the moment generating function, namely

$$g(t) = \ln M(t) = \mu t + \frac{1}{2}\sigma^2 t^2$$

Since this is a quadratic polynomial in  $t$ , only the first two cumulants are nonzero, namely the mean  $\mu$  and the variance  $\sigma^2$ .

## Stein operator and class

Within Stein's method the Stein operator and class of a random variable  $\mathbf{X} \sim \mathcal{N}(\mu, \sigma^2)$  are  $\mathcal{A}f(x) = \sigma^2 f'(x) - (x - \mu)f(x)$  and  $\mathcal{F}$  the class of all absolutely continuous functions  $f : \mathbb{R} \rightarrow \mathbb{R}$  such that  $\mathbb{E}[|f'(X)|] < \infty$ .

## Zero-variance limit

In the limit when  $\sigma$  tends to zero, the probability density  $f(x)$  eventually tends to zero at any  $x \neq \mu$ , but grows without limit if  $x = \mu$ , while its integral remains equal to 1. Therefore, the normal distribution cannot be defined as an ordinary function when  $\sigma = 0$ .

However, one can define the normal distribution with zero variance as a generalized function; specifically, as Dirac's "delta function"  $\delta$  translated by the mean  $\mu$ , that is  $f(x) = \delta(x - \mu)$ . Its CDF is then the Heaviside step function translated by the mean  $\mu$ , namely

$$F(x) = \begin{cases} 0 & \text{if } x < \mu \\ 1 & \text{if } x \geq \mu \end{cases}$$

## Maximum entropy

Of all probability distributions over the reals with a specified mean  $\mu$  and variance  $\sigma^2$ , the normal distribution  $\mathcal{N}(\mu, \sigma^2)$  is the one with maximum entropy.<sup>[28]</sup> If  $\mathbf{X}$  is a continuous random variable with probability density  $f(x)$ , then the entropy of  $\mathbf{X}$  is defined as<sup>[29][30][31]</sup>

$$H(\mathbf{X}) = - \int_{-\infty}^{\infty} f(x) \log f(x) dx$$

where  $f(x) \log f(x)$  is understood to be zero whenever  $f(x) = 0$ . This functional can be maximized, subject to the constraints that the distribution is properly normalized and has a specified variance, by using variational calculus. A function with two Lagrange multipliers is defined:

$$L = \int_{-\infty}^{\infty} f(x) \ln(f(x)) dx - \lambda_0 \left( 1 - \int_{-\infty}^{\infty} f(x) dx \right) - \lambda \left( \sigma^2 - \int_{-\infty}^{\infty} f(x)(x - \mu)^2 dx \right)$$

where  $f(x)$  is, for now, regarded as some density function with mean  $\mu$  and standard deviation  $\sigma$ .

At maximum entropy, a small variation  $\delta f(x)$  about  $f(x)$  will produce a variation  $\delta L$  about  $L$  which is equal to 0:

$$0 = \delta L = \int_{-\infty}^{\infty} \delta f(x) (\ln(f(x)) + 1 + \lambda_0 + \lambda(x - \mu)^2) dx$$

Since this must hold for any small  $\delta f(x)$ , the term in brackets must be zero, and solving for  $f(x)$  yields:

$$f(x) = e^{-\lambda_0 - 1 - \lambda(x - \mu)^2}$$

Using the constraint equations to solve for  $\lambda_0$  and  $\lambda$  yields the density of the normal distribution:

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The entropy of a normal distribution is equal to

$$H(x) = \frac{1}{2} (1 + \log(2\sigma^2\pi))$$

## Operations on normal deviates

The family of normal distributions is closed under linear transformations: if  $\mathbf{X}$  is normally distributed with mean  $\mu$  and standard deviation  $\sigma$ , then the variable  $\mathbf{Y} = \mathbf{a}\mathbf{X} + \mathbf{b}$ , for any real numbers  $\mathbf{a}$  and  $\mathbf{b}$ , is also normally distributed, with mean  $\mathbf{a}\mu + \mathbf{b}$  and standard deviation  $|\mathbf{a}|\sigma$ .

Also if  $\mathbf{X}_1$  and  $\mathbf{X}_2$  are two independent normal random variables, with means  $\mu_1, \mu_2$  and standard deviations  $\sigma_1, \sigma_2$ , then their sum  $\mathbf{X}_1 + \mathbf{X}_2$  will also be normally distributed,<sup>[proof]</sup> with mean  $\mu_1 + \mu_2$  and variance  $\sigma_1^2 + \sigma_2^2$ .

In particular, if  $\mathbf{X}$  and  $\mathbf{Y}$  are independent normal deviates with zero mean and variance  $\sigma^2$ , then  $\mathbf{X} + \mathbf{Y}$  and  $\mathbf{X} - \mathbf{Y}$  are also independent and normally distributed, with zero mean and variance  $2\sigma^2$ . This is a special case of the polarization identity.<sup>[32]</sup>

Also, if  $\mathbf{X}_1, \mathbf{X}_2$  are two independent normal deviates with mean  $\mu$  and deviation  $\sigma$ , and  $\mathbf{a}, \mathbf{b}$  are arbitrary real numbers, then the variable

$$\mathbf{X}_3 = \frac{\mathbf{a}\mathbf{X}_1 + \mathbf{b}\mathbf{X}_2 - (\mathbf{a} + \mathbf{b})\mu}{\sqrt{\mathbf{a}^2 + \mathbf{b}^2}} + \mu$$

is also normally distributed with mean  $\mu$  and deviation  $\sigma$ . It follows that the normal distribution is stable (with exponent  $\alpha = 2$ ).

More generally, any linear combination of independent normal deviates is a normal deviate.

## Infinite divisibility and Cramér's theorem

For any positive integer  $\mathbf{n}$ , any normal distribution with mean  $\mu$  and variance  $\sigma^2$  is the distribution of the sum of  $\mathbf{n}$  independent normal deviates, each with mean  $\frac{\mu}{\mathbf{n}}$  and variance  $\frac{\sigma^2}{\mathbf{n}}$ . This property is called infinite divisibility.<sup>[33]</sup>

Conversely, if  $\mathbf{X}_1$  and  $\mathbf{X}_2$  are independent random variables and their sum  $\mathbf{X}_1 + \mathbf{X}_2$  has a normal distribution, then both  $\mathbf{X}_1$  and  $\mathbf{X}_2$  must be normal deviates.<sup>[34]</sup>

This result is known as Cramér's decomposition theorem, and is equivalent to saying that the convolution of two distributions is normal if and only if both are normal. Cramér's theorem implies that a linear combination of independent non-Gaussian variables will never have an exactly normal distribution, although it may approach it arbitrarily closely.<sup>[35]</sup>

## Bernstein's theorem

Bernstein's theorem states that if  $\mathbf{X}$  and  $\mathbf{Y}$  are independent and  $\mathbf{X} + \mathbf{Y}$  and  $\mathbf{X} - \mathbf{Y}$  are also independent, then both  $X$  and  $Y$  must necessarily have normal distributions.<sup>[36][37]</sup>

More generally, if  $\mathbf{X}_1, \dots, \mathbf{X}_n$  are independent random variables, then two distinct linear combinations  $\sum a_k \mathbf{X}_k$  and  $\sum b_k \mathbf{X}_k$  will be independent if and only if all  $\mathbf{X}_k$  are normal and  $\sum a_k b_k \sigma_k^2 = 0$ , where  $\sigma_k^2$  denotes the variance of  $\mathbf{X}_k$ .<sup>[36]</sup>

## Other properties

1. If the characteristic function  $\phi_X$  of some random variable  $\mathbf{X}$  is of the form  $\phi_X(t) = \exp^{Q(t)}$ , where  $Q(t)$  is a polynomial, then the **Marcinkiewicz theorem** (named after Józef Marcinkiewicz) asserts that  $Q$  can be at most a quadratic polynomial, and therefore  $\mathbf{X}$  is a normal random variable.<sup>[35]</sup> The consequence of this result is that the normal distribution is the only distribution with a finite number (two) of non-zero cumulants.
2. If  $\mathbf{X}$  and  $\mathbf{Y}$  are jointly normal and uncorrelated, then they are independent. The requirement that  $\mathbf{X}$  and  $\mathbf{Y}$  should be *jointly* normal is essential; without it the property does not hold.<sup>[38][39][proof]</sup> For non-normal random variables uncorrelatedness does not imply independence.
3. The Kullback–Leibler divergence of one normal distribution  $\mathbf{X}_1 \sim N(\mu_1, \sigma_1^2)$  from another  $\mathbf{X}_2 \sim N(\mu_2, \sigma_2^2)$  is given by:<sup>[40]</sup>

$$D_{\text{KL}}(\mathbf{X}_1 \parallel \mathbf{X}_2) = \frac{(\mu_1 - \mu_2)^2}{2\sigma_2^2} + \frac{1}{2} \left( \frac{\sigma_1^2}{\sigma_2^2} - 1 - \ln \frac{\sigma_1^2}{\sigma_2^2} \right)$$

The Hellinger distance between the same distributions is equal to

$$H^2(\mathbf{X}_1, \mathbf{X}_2) = 1 - \sqrt{\frac{2\sigma_1\sigma_2}{\sigma_1^2 + \sigma_2^2}} e^{-\frac{1}{4} \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}}$$

4. The Fisher information matrix for a normal distribution is diagonal and takes the form

$$\mathcal{I} = \begin{pmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{1}{2\sigma^4} \end{pmatrix}$$

5. The conjugate prior of the mean of a normal distribution is another normal distribution.<sup>[41]</sup> Specifically, if  $\mathbf{x}_1, \dots, \mathbf{x}_n$  are iid  $\sim N(\mu, \sigma^2)$  and the prior is  $\mu \sim N(\mu_0, \sigma_0^2)$ , then the posterior distribution for the estimator of  $\mu$  will be

$$\mu \mid \mathbf{x}_1, \dots, \mathbf{x}_n \sim \mathcal{N} \left( \frac{\frac{\sigma^2}{n} \mu_0 + \sigma_0^2 \bar{x}}{\frac{\sigma^2}{n} + \sigma_0^2}, \left( \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \right)^{-1} \right)$$

6. The family of normal distributions not only forms an exponential family (EF), but in fact forms a natural exponential family (NEF) with quadratic variance function (NEF-QVF). Many properties of normal distributions generalize to properties of NEF-QVF distributions, NEF distributions, or EF distributions generally. NEF-QVF distributions comprises 6 families, including Poisson, Gamma, binomial, and negative binomial distributions, while many of the common families studied in probability and statistics are NEF or EF.
7. In information geometry, the family of normal distributions forms a statistical manifold with constant curvature  $-1$ . The same family is flat with respect to the  $(\pm 1)$ -connections  $\nabla^{(e)}$  and  $\nabla^{(m)}$ .<sup>[42]</sup>

## Related distributions

## Central limit theorem

The central limit theorem states that under certain (fairly common) conditions, the sum of many random variables will have an approximately normal distribution. More specifically, where  $\mathbf{X}_1, \dots, \mathbf{X}_n$  are independent and identically distributed random variables with the same arbitrary distribution, zero mean, and variance  $\sigma^2$  and  $\mathbf{Z}$  is their mean scaled by  $\sqrt{n}$

$$\mathbf{Z} = \sqrt{n} \left( \frac{1}{n} \sum_{i=1}^n \mathbf{X}_i \right)$$

Then, as  $n$  increases, the probability distribution of  $\mathbf{Z}$  will tend to the normal distribution with zero mean and variance  $\sigma^2$ .

The theorem can be extended to variables ( $\mathbf{X}_i$ ) that are not independent and/or not identically distributed if certain constraints are placed on the degree of dependence and the moments of the distributions.

Many test statistics, scores, and estimators encountered in practice contain sums of certain random variables in them, and even more estimators can be represented as sums of random variables through the use of influence functions. The central limit theorem implies that those statistical parameters will have asymptotically normal distributions.

The central limit theorem also implies that certain distributions can be approximated by the normal distribution, for example:

- The binomial distribution  $B(n, p)$  is approximately normal with mean  $np$  and variance  $np(1 - p)$  for large  $n$  and for  $p$  not too close to 0 or 1.
- The Poisson distribution with parameter  $\lambda$  is approximately normal with mean  $\lambda$  and variance  $\lambda$ , for large values of  $\lambda$ .<sup>[43]</sup>
- The chi-squared distribution  $\chi^2(k)$  is approximately normal with mean  $k$  and variance  $2k$ , for large  $k$ .
- The Student's t-distribution  $t(\nu)$  is approximately normal with mean 0 and variance 1 when  $\nu$  is large.

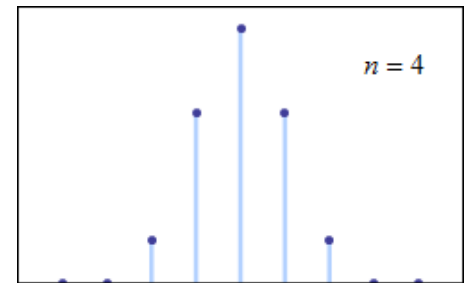
Whether these approximations are sufficiently accurate depends on the purpose for which they are needed, and the rate of convergence to the normal distribution. It is typically the case that such approximations are less accurate in the tails of the distribution.

A general upper bound for the approximation error in the central limit theorem is given by the Berry–Esseen theorem, improvements of the approximation are given by the Edgeworth expansions.

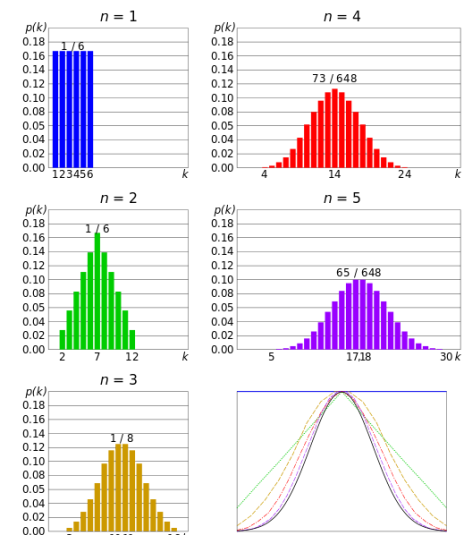
## Operations on a single random variable

If  $X$  is distributed normally with mean  $\mu$  and variance  $\sigma^2$ , then

- The exponential of  $X$  is distributed log-normally:  $e^X \sim \ln(N(\mu, \sigma^2))$ .
- The absolute value of  $X$  has folded normal distribution:  $|X| \sim N_f(\mu, \sigma^2)$ . If  $\mu = 0$  this is known as the half-normal distribution.
- The absolute value of normalized residuals,  $|X - \mu|/\sigma$ , has chi distribution with one degree of freedom:  $|X - \mu|/\sigma \sim \chi_1$ .



As the number of discrete events increases, the function begins to resemble a normal distribution



Comparison of probability density functions,  $p(k)$  for the sum of  $n$  fair 6-sided dice to show their convergence to a normal distribution with increasing  $n$ , in accordance to the central limit theorem. In the bottom-right graph, smoothed profiles of the previous graphs are rescaled, superimposed and compared with a normal distribution (black curve).

- The square of  $X/\sigma$  has the noncentral chi-squared distribution with one degree of freedom:  $X^2/\sigma^2 \sim \chi_1^2(\mu^2/\sigma^2)$ . If  $\mu = 0$ , the distribution is called simply chi-squared.
- The distribution of the variable  $X$  restricted to an interval  $[a, b]$  is called the truncated normal distribution.
- $(X - \mu)^{-2}$  has a Lévy distribution with location 0 and scale  $\sigma^{-2}$ .

## Combination of two independent random variables

If  $X_1$  and  $X_2$  are two independent standard normal random variables with mean 0 and variance 1, then

- Their sum and difference is distributed normally with mean zero and variance two:  $X_1 \pm X_2 \sim N(0, 2)$ .
- Their product  $Z = X_1 X_2$  follows the "product-normal" distribution<sup>[44]</sup> with density function  $f_Z(z) = \pi^{-1} K_0(|z|)$  where  $K_0$  is the modified Bessel function of the second kind. This distribution is symmetric around zero, unbounded at  $z = 0$ , and has the characteristic function  $\phi_Z(t) = (1 + t^2)^{-1/2}$ .
- Their ratio follows the standard Cauchy distribution:  $X_1/X_2 \sim \text{Cauchy}(0, 1)$ .
- Their Euclidean norm  $\sqrt{X_1^2 + X_2^2}$  has the Rayleigh distribution.

## Combination of two or more independent random variables

- If  $X_1, X_2, \dots, X_n$  are independent standard normal random variables, then the sum of their squares has the chi-squared distribution with  $n$  degrees of freedom

$$X_1^2 + \dots + X_n^2 \sim \chi_n^2.$$

- If  $X_1, X_2, \dots, X_n$  are independent normally distributed random variables with means  $\mu$  and variances  $\sigma^2$ , then their sample mean is independent from the sample standard deviation,<sup>[45]</sup> which can be demonstrated using Basu's theorem or Cochran's theorem.<sup>[46]</sup> The ratio of these two quantities will have the Student's t-distribution with  $n - 1$  degrees of freedom:

$$t = \frac{\bar{X} - \mu}{S/\sqrt{n}} = \frac{\frac{1}{n}(X_1 + \dots + X_n) - \mu}{\sqrt{\frac{1}{n(n-1)} [(X_1 - \bar{X})^2 + \dots + (X_n - \bar{X})^2]}} \sim t_{n-1}.$$

- If  $X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_m$  are independent standard normal random variables, then the ratio of their normalized sums of squares will have the F-distribution with  $(n, m)$  degrees of freedom:<sup>[47]</sup>

$$F = \frac{(X_1^2 + X_2^2 + \dots + X_n^2) / n}{(Y_1^2 + Y_2^2 + \dots + Y_m^2) / m} \sim F_{n,m}.$$

## Operations on the density function

The split normal distribution is most directly defined in terms of joining scaled sections of the density functions of different normal distributions and rescaling the density to integrate to one. The truncated normal distribution results from rescaling a section of a single density function.

## Extensions

The notion of normal distribution, being one of the most important distributions in probability theory, has been extended far beyond the standard framework of the univariate (that is one-dimensional) case (Case 1). All these extensions are also called *normal* or *Gaussian* laws, so a certain ambiguity in names exists.

- The multivariate normal distribution describes the Gaussian law in the  $k$ -dimensional Euclidean space. A vector  $X \in \mathbf{R}^k$  is multivariate-normally distributed if any linear combination of its components  $\sum_{j=1}^k a_j X_j$  has a (univariate) normal distribution. The variance of  $X$  is a  $k \times k$  symmetric positive-definite matrix  $V$ . The multivariate normal distribution is a special case of the elliptical distributions. As such, its iso-density loci in the  $k = 2$  case are ellipses and in the case of arbitrary  $k$  are ellipsoids.
- Rectified Gaussian distribution a rectified version of normal distribution with all the negative elements reset to 0
- Complex normal distribution deals with the complex normal vectors. A complex vector  $X \in \mathbf{C}^k$  is said to be normal if both its real and imaginary components jointly possess a  $2k$ -dimensional multivariate normal distribution. The variance-covariance structure of  $X$  is described by two matrices: the *variance* matrix  $\Gamma$ , and the *relation* matrix  $C$ .
- Matrix normal distribution describes the case of normally distributed matrices.
- Gaussian processes are the normally distributed stochastic processes. These can be viewed as elements of some infinite-dimensional Hilbert space  $H$ , and thus are the analogues of multivariate normal vectors for the case  $k = \infty$ . A random element  $h \in H$  is said to be normal if for any constant  $a \in H$  the scalar product  $(a, h)$  has a (univariate) normal distribution. The variance structure of such Gaussian random element can be described in terms of the linear *covariance operator*  $K: H \rightarrow H$ . Several Gaussian processes became popular enough to have their own names:
  - Brownian motion,
  - Brownian bridge,
  - Ornstein–Uhlenbeck process.
- Gaussian q-distribution is an abstract mathematical construction that represents a "q-analogue" of the normal distribution.
- the q-Gaussian is an analogue of the Gaussian distribution, in the sense that it maximises the Tsallis entropy, and is one type of Tsallis distribution. Note that this distribution is different from the Gaussian q-distribution above.

A random variable  $X$  has a two-piece normal distribution if it has a distribution

$$\begin{aligned} f_X(x) &= N(\mu, \sigma_1^2) \text{ if } x \leq \mu \\ f_X(x) &= N(\mu, \sigma_2^2) \text{ if } x \geq \mu \end{aligned}$$

where  $\mu$  is the mean and  $\sigma_1$  and  $\sigma_2$  are the standard deviations of the distribution to the left and right of the mean respectively.

The mean, variance and third central moment of this distribution have been determined<sup>[48]</sup>

$$\begin{aligned} E(X) &= \mu + \sqrt{\frac{2}{\pi}}(\sigma_2 - \sigma_1) \\ V(X) &= \left(1 - \frac{2}{\pi}\right)(\sigma_2 - \sigma_1)^2 + \sigma_1\sigma_2 \\ T(X) &= \sqrt{\frac{2}{\pi}}(\sigma_2 - \sigma_1) \left[ \left(\frac{4}{\pi} - 1\right)(\sigma_2 - \sigma_1)^2 + \sigma_1\sigma_2 \right] \end{aligned}$$

where  $E(X)$ ,  $V(X)$  and  $T(X)$  are the mean, variance, and third central moment respectively.

One of the main practical uses of the Gaussian law is to model the empirical distributions of many different random variables encountered in practice. In such case a possible extension would be a richer family of distributions, having more than two parameters and therefore being able to fit the empirical distribution more accurately. The examples of such extensions are:

- Pearson distribution — a four-parameter family of probability distributions that extend the normal law to include different skewness and kurtosis values.

- The generalized normal distribution, also known as the exponential power distribution, allows for distribution tails with thicker or thinner asymptotic behaviors.

## Statistical inference

### Estimation of parameters

It is often the case that we do not know the parameters of the normal distribution, but instead want to estimate them. That is, having a sample  $(x_1, \dots, x_n)$  from a normal  $N(\mu, \sigma^2)$  population we would like to learn the approximate values of parameters  $\mu$  and  $\sigma^2$ . The standard approach to this problem is the maximum likelihood method, which requires maximization of the *log-likelihood function*:

$$\ln \mathcal{L}(\mu, \sigma^2) = \sum_{i=1}^n \ln f(x_i | \mu, \sigma^2) = -\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2.$$

Taking derivatives with respect to  $\mu$  and  $\sigma^2$  and solving the resulting system of first order conditions yields the *maximum likelihood estimates*:

$$\hat{\mu} = \bar{x} \equiv \frac{1}{n} \sum_{i=1}^n x_i, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2.$$

### Sample mean

Estimator  $\hat{\mu}$  is called the *sample mean*, since it is the arithmetic mean of all observations. The statistic  $\bar{x}$  is complete and sufficient for  $\mu$ , and therefore by the Lehmann–Scheffé theorem,  $\hat{\mu}$  is the uniformly minimum variance unbiased (UMVU) estimator.<sup>[49]</sup> In finite samples it is distributed normally:

$$\hat{\mu} \sim \mathcal{N}(\mu, \sigma^2/n).$$

The variance of this estimator is equal to the  $\mu\mu$ -element of the inverse Fisher information matrix  $\mathcal{I}^{-1}$ . This implies that the estimator is finite-sample efficient. Of practical importance is the fact that the standard error of  $\hat{\mu}$  is proportional to  $1/\sqrt{n}$ , that is, if one wishes to decrease the standard error by a factor of 10, one must increase the number of points in the sample by a factor of 100. This fact is widely used in determining sample sizes for opinion polls and the number of trials in Monte Carlo simulations.

From the standpoint of the asymptotic theory,  $\hat{\mu}$  is consistent, that is, it converges in probability to  $\mu$  as  $n \rightarrow \infty$ . The estimator is also asymptotically normal, which is a simple corollary of the fact that it is normal in finite samples:

$$\sqrt{n}(\hat{\mu} - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2).$$

### Sample variance

The estimator  $\hat{\sigma}^2$  is called the *sample variance*, since it is the variance of the sample  $((x_1, \dots, x_n))$ . In practice, another estimator is often used instead of the  $\hat{\sigma}^2$ . This other estimator is denoted  $s^2$ , and is also called the *sample variance*, which represents a certain ambiguity in terminology; its square root  $s$  is called the *sample standard deviation*. The estimator  $s^2$  differs from  $\hat{\sigma}^2$  by having  $(n - 1)$  instead of  $n$  in the denominator (the so-called Bessel's correction):

$$s^2 = \frac{n}{n-1} \hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2.$$

The difference between  $\mathbf{s}^2$  and  $\hat{\sigma}^2$  becomes negligibly small for large  $n$ 's. In finite samples however, the motivation behind the use of  $\mathbf{s}^2$  is that it is an unbiased estimator of the underlying parameter  $\sigma^2$ , whereas  $\hat{\sigma}^2$  is biased. Also, by the Lehmann–Scheffé theorem the estimator  $\mathbf{s}^2$  is uniformly minimum variance unbiased (UMVU),<sup>[49]</sup> which makes it the "best" estimator among all unbiased ones. However it can be shown that the biased estimator  $\hat{\sigma}^2$  is "better" than the  $\mathbf{s}^2$  in terms of the mean squared error (MSE) criterion. In finite samples both  $\mathbf{s}^2$  and  $\hat{\sigma}^2$  have scaled chi-squared distribution with  $(n - 1)$  degrees of freedom:

$$\mathbf{s}^2 \sim \frac{\sigma^2}{n-1} \cdot \chi_{n-1}^2, \quad \hat{\sigma}^2 \sim \frac{\sigma^2}{n} \cdot \chi_{n-1}^2.$$

The first of these expressions shows that the variance of  $\mathbf{s}^2$  is equal to  $2\sigma^4/(n-1)$ , which is slightly greater than the  $\sigma\sigma$ -element of the inverse Fisher information matrix  $\mathcal{I}^{-1}$ . Thus,  $\mathbf{s}^2$  is not an efficient estimator for  $\sigma^2$ , and moreover, since  $\mathbf{s}^2$  is UMVU, we can conclude that the finite-sample efficient estimator for  $\sigma^2$  does not exist.

Applying the asymptotic theory, both estimators  $\mathbf{s}^2$  and  $\hat{\sigma}^2$  are consistent, that is they converge in probability to  $\sigma^2$  as the sample size  $n \rightarrow \infty$ . The two estimators are also both asymptotically normal:

$$\sqrt{n}(\hat{\sigma}^2 - \sigma^2) \simeq \sqrt{n}(\mathbf{s}^2 - \sigma^2) \xrightarrow{d} \mathcal{N}(0, 2\sigma^4).$$

In particular, both estimators are asymptotically efficient for  $\sigma^2$ .

## Confidence intervals

By Cochran's theorem, for normal distributions the sample mean  $\hat{\mu}$  and the sample variance  $\mathbf{s}^2$  are independent, which means there can be no gain in considering their joint distribution. There is also a converse theorem: if in a sample the sample mean and sample variance are independent, then the sample must have come from the normal distribution. The independence between  $\hat{\mu}$  and  $\mathbf{s}^2$  can be employed to construct the so-called *t-statistic*:

$$t = \frac{\hat{\mu} - \mu}{\mathbf{s}/\sqrt{n}} = \frac{\bar{x} - \mu}{\sqrt{\frac{1}{n(n-1)} \sum (x_i - \bar{x})^2}} \sim t_{n-1}$$

This quantity  $t$  has the Student's t-distribution with  $(n - 1)$  degrees of freedom, and it is an ancillary statistic (independent of the value of the parameters). Inverting the distribution of this  $t$ -statistics will allow us to construct the confidence interval for  $\mu$ ;<sup>[50]</sup> similarly, inverting the  $\chi^2$  distribution of the statistic  $\mathbf{s}^2$  will give us the confidence interval for  $\sigma^2$ .<sup>[51]</sup>

$$\begin{aligned} \mu &\in \left[ \hat{\mu} - t_{n-1, 1-\alpha/2} \frac{1}{\sqrt{n}} \mathbf{s}, \hat{\mu} + t_{n-1, 1-\alpha/2} \frac{1}{\sqrt{n}} \mathbf{s} \right] \approx \left[ \hat{\mu} - |z_{\alpha/2}| \frac{1}{\sqrt{n}} \mathbf{s}, \hat{\mu} + |z_{\alpha/2}| \frac{1}{\sqrt{n}} \mathbf{s} \right], \\ \sigma^2 &\in \left[ \frac{(n-1)\mathbf{s}^2}{\chi_{n-1, 1-\alpha/2}^2}, \frac{(n-1)\mathbf{s}^2}{\chi_{n-1, \alpha/2}^2} \right] \approx \left[ \mathbf{s}^2 - |z_{\alpha/2}| \frac{\sqrt{2}}{\sqrt{n}} \mathbf{s}^2, \mathbf{s}^2 + |z_{\alpha/2}| \frac{\sqrt{2}}{\sqrt{n}} \mathbf{s}^2 \right], \end{aligned}$$

where  $t_{k,p}$  and  $\chi_{k,p}^2$  are the  $p$ th quantiles of the  $t$ - and  $\chi^2$ -distributions respectively. These confidence intervals are of the confidence level  $1 - \alpha$ , meaning that the true values  $\mu$  and  $\sigma^2$  fall outside of these intervals with probability (or significance level)  $\alpha$ . In practice people usually take  $\alpha = 5\%$ , resulting in the 95% confidence intervals. The approximate formulas in the display above were derived from the asymptotic distributions of  $\hat{\mu}$  and  $\mathbf{s}^2$ . The approximate formulas become valid for large values of  $n$ , and are more convenient for the manual calculation since the standard normal quantiles  $z_{\alpha/2}$  do not depend on  $n$ . In particular, the most popular value of  $\alpha = 5\%$ , results in  $|z_{0.025}| = 1.96$ .

## Normality tests

Normality tests assess the likelihood that the given data set  $\{x_1, \dots, x_n\}$  comes from a normal distribution. Typically the null hypothesis  $H_0$  is that the observations are distributed normally with unspecified mean  $\mu$  and variance  $\sigma^2$ , versus the alternative  $H_a$  that the distribution is arbitrary. Many tests (over 40) have been devised for this problem, the more prominent of them are outlined below:

- **"Visual" tests** are more intuitively appealing but subjective at the same time, as they rely on informal human judgement to accept or reject the null hypothesis.
  - Q-Q plot— is a plot of the sorted values from the data set against the expected values of the corresponding quantiles from the standard normal distribution. That is, it's a plot of point of the form  $(\Phi^{-1}(p_k), x_{(k)})$ , where plotting points  $p_k$  are equal to  $p_k = (k - \alpha)/(n + 1 - 2\alpha)$  and  $\alpha$  is an adjustment constant, which can be anything between 0 and 1. If the null hypothesis is true, the plotted points should approximately lie on a straight line.
  - P-P plot— similar to the Q-Q plot, but used much less frequently. This method consists of plotting the points  $(\Phi(z_{(k)}), p_k)$ , where  $z_{(k)} = (x_{(k)} - \hat{\mu})/\hat{\sigma}$ . For normally distributed data this plot should lie on a 45° line between (0, 0) and (1, 1).
  - Shapiro-Wilk test employs the fact that the line in the Q-Q plot has the slope of  $\sigma$ . The test compares the least squares estimate of that slope with the value of the sample variance, and rejects the null hypothesis if these two quantities differ significantly.
  - Normal probability plot (rankit plot)
- **Moment tests:**
  - D'Agostino's K-squared test
  - Jarque–Bera test
- **Empirical distribution function tests:**
  - Lilliefors test (an adaptation of the Kolmogorov–Smirnov test)
  - Anderson–Darling test

## Bayesian analysis of the normal distribution

Bayesian analysis of normally distributed data is complicated by the many different possibilities that may be considered:

- Either the mean, or the variance, or neither, may be considered a fixed quantity.
- When the variance is unknown, analysis may be done directly in terms of the variance, or in terms of the precision, the reciprocal of the variance. The reason for expressing the formulas in terms of precision is that the analysis of most cases is simplified.
- Both univariate and multivariate cases need to be considered.
- Either conjugate or improper prior distributions may be placed on the unknown variables.
- An additional set of cases occurs in Bayesian linear regression, where in the basic model the data is assumed to be normally distributed, and normal priors are placed on the regression coefficients. The resulting analysis is similar to the basic cases of independent identically distributed data.

The formulas for the non-linear-regression cases are summarized in the conjugate prior article.

## Sum of two quadratics

### Scalar form

The following auxiliary formula is useful for simplifying the posterior update equations, which otherwise become fairly tedious.

$$a(x - y)^2 + b(x - z)^2 = (a + b) \left( x - \frac{ay + bz}{a + b} \right)^2 + \frac{ab}{a + b} (y - z)^2$$

This equation rewrites the sum of two quadratics in  $x$  by expanding the squares, grouping the terms in  $x$ , and completing the square. Note the following about the complex constant factors attached to some of the terms:

1. The factor  $\frac{ay + bz}{a + b}$  has the form of a weighted average of  $y$  and  $z$ .
2.  $\frac{ab}{a + b} = \frac{1}{\frac{1}{a} + \frac{1}{b}} = (a^{-1} + b^{-1})^{-1}$ . This shows that this factor can be thought of as resulting from a situation where the reciprocals of quantities  $a$  and  $b$  add directly, so to combine  $a$  and  $b$  themselves, it's necessary to reciprocate, add, and reciprocate the result again to get back into the original units. This is exactly the sort of operation performed by the harmonic mean, so it is not surprising that  $\frac{ab}{a + b}$  is one-half the harmonic mean of  $a$  and  $b$ .

## Vector form

A similar formula can be written for the sum of two vector quadratics: If  $\mathbf{x}$ ,  $\mathbf{y}$ ,  $\mathbf{z}$  are vectors of length  $k$ , and  $\mathbf{A}$  and  $\mathbf{B}$  are symmetric, invertible matrices of size  $k \times k$ , then

$$\begin{aligned} & (\mathbf{y} - \mathbf{x})' \mathbf{A} (\mathbf{y} - \mathbf{x}) + (\mathbf{x} - \mathbf{z})' \mathbf{B} (\mathbf{x} - \mathbf{z}) \\ &= (\mathbf{x} - \mathbf{c})' (\mathbf{A} + \mathbf{B}) (\mathbf{x} - \mathbf{c}) + (\mathbf{y} - \mathbf{z})' (\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1} (\mathbf{y} - \mathbf{z}) \end{aligned}$$

where

$$\mathbf{c} = (\mathbf{A} + \mathbf{B})^{-1} (\mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{z})$$

Note that the form  $\mathbf{x}' \mathbf{A} \mathbf{x}$  is called a quadratic form and is a scalar:

$$\mathbf{x}' \mathbf{A} \mathbf{x} = \sum_{i,j} a_{ij} x_i x_j$$

In other words, it sums up all possible combinations of products of pairs of elements from  $\mathbf{x}$ , with a separate coefficient for each. In addition, since  $x_i x_j = x_j x_i$ , only the sum  $a_{ij} + a_{ji}$  matters for any off-diagonal elements of  $\mathbf{A}$ , and there is no loss of generality in assuming that  $\mathbf{A}$  is symmetric. Furthermore, if  $\mathbf{A}$  is symmetric, then the form  $\mathbf{x}' \mathbf{A} \mathbf{y} = \mathbf{y}' \mathbf{A} \mathbf{x}$ .

## Sum of differences from the mean

Another useful formula is as follows:

$$\sum_{i=1}^n (x_i - \mu)^2 = \sum_{i=1}^n (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2$$

$$\text{where } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i.$$

## With known variance

For a set of i.i.d. normally distributed data points  $\mathbf{X}$  of size  $n$  where each individual point  $x$  follows  $x \sim \mathcal{N}(\mu, \sigma^2)$  with known variance  $\sigma^2$ , the conjugate prior distribution is also normally distributed.

This can be shown more easily by rewriting the variance as the precision, i.e. using  $\tau = 1/\sigma^2$ . Then if  $x \sim \mathcal{N}(\mu, 1/\tau)$  and  $\mu \sim \mathcal{N}(\mu_0, 1/\tau_0)$ , we proceed as follows.

First, the likelihood function is (using the formula above for the sum of differences from the mean):

$$\begin{aligned} p(\mathbf{X} \mid \mu, \tau) &= \prod_{i=1}^n \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{1}{2}\tau(x_i - \mu)^2\right) \\ &= \left(\frac{\tau}{2\pi}\right)^{n/2} \exp\left(-\frac{1}{2}\tau \sum_{i=1}^n (x_i - \mu)^2\right) \\ &= \left(\frac{\tau}{2\pi}\right)^{n/2} \exp\left[-\frac{1}{2}\tau \left(\sum_{i=1}^n (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2\right)\right]. \end{aligned}$$

Then, we proceed as follows:

$$\begin{aligned} p(\mu \mid \mathbf{X}) &\propto p(\mathbf{X} \mid \mu)p(\mu) \\ &= \left(\frac{\tau}{2\pi}\right)^{n/2} \exp\left[-\frac{1}{2}\tau \left(\sum_{i=1}^n (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2\right)\right] \sqrt{\frac{\tau_0}{2\pi}} \exp\left(-\frac{1}{2}\tau_0(\mu - \mu_0)^2\right) \\ &\propto \exp\left(-\frac{1}{2} \left(\tau \left(\sum_{i=1}^n (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2\right) + \tau_0(\mu - \mu_0)^2\right)\right) \\ &\propto \exp\left(-\frac{1}{2} (n\tau(\bar{x} - \mu)^2 + \tau_0(\mu - \mu_0)^2)\right) \\ &= \exp\left(-\frac{1}{2}(n\tau + \tau_0) \left(\mu - \frac{n\tau\bar{x} + \tau_0\mu_0}{n\tau + \tau_0}\right)^2 + \frac{n\tau\tau_0}{n\tau + \tau_0} (\bar{x} - \mu_0)^2\right) \\ &\propto \exp\left(-\frac{1}{2}(n\tau + \tau_0) \left(\mu - \frac{n\tau\bar{x} + \tau_0\mu_0}{n\tau + \tau_0}\right)^2\right) \end{aligned}$$

In the above derivation, we used the formula above for the sum of two quadratics and eliminated all constant factors not involving  $\mu$ . The result is the kernel of a normal distribution, with mean  $\frac{n\tau\bar{x} + \tau_0\mu_0}{n\tau + \tau_0}$  and precision  $n\tau + \tau_0$ , i.e.

$$p(\mu \mid \mathbf{X}) \sim \mathcal{N}\left(\frac{n\tau\bar{x} + \tau_0\mu_0}{n\tau + \tau_0}, \frac{1}{n\tau + \tau_0}\right)$$

This can be written as a set of Bayesian update equations for the posterior parameters in terms of the prior parameters:

$$\begin{aligned} \tau'_0 &= \tau_0 + n\tau \\ \mu'_0 &= \frac{n\tau\bar{x} + \tau_0\mu_0}{n\tau + \tau_0} \\ \bar{x} &= \frac{1}{n} \sum_{i=1}^n x_i \end{aligned}$$

That is, to combine  $n$  data points with total precision of  $n\tau$  (or equivalently, total variance of  $n/\sigma^2$ ) and mean of values  $\bar{x}$ , derive a new total precision simply by adding the total precision of the data to the prior total precision, and form a new mean through a *precision-weighted average*, i.e. a weighted average of the data mean and the prior mean, each weighted by the associated total precision. This makes logical sense if the precision is thought of as indicating the certainty of the observations: In the distribution of the posterior mean, each of the input components is weighted by its certainty, and the certainty of this distribution is the sum of the individual certainties. (For the intuition of this, compare the expression "the whole is (or is not) greater than the sum of its parts". In addition, consider that the knowledge of the posterior comes from a combination of the knowledge of the prior and likelihood, so it makes sense that we are more certain of it than of either of its components.)

The above formula reveals why it is more convenient to do Bayesian analysis of conjugate priors for the normal distribution in terms of the precision. The posterior precision is simply the sum of the prior and likelihood precisions, and the posterior mean is computed through a precision-weighted average, as described above. The same formulas can be written in terms of variance by reciprocating all the precisions, yielding the more ugly formulas

$$\begin{aligned}\sigma_0'^2 &= \frac{1}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}} \\ \mu_0' &= \frac{\frac{n\bar{x}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}} \\ \bar{x} &= \frac{1}{n} \sum_{i=1}^n x_i\end{aligned}$$

### With known mean

For a set of i.i.d. normally distributed data points  $\mathbf{X}$  of size  $n$  where each individual point  $x$  follows  $\mathbf{x} \sim \mathcal{N}(\mu, \sigma^2)$  with known mean  $\mu$ , the conjugate prior of the variance has an inverse gamma distribution or a scaled inverse chi-squared distribution. The two are equivalent except for having different parameterizations. Although the inverse gamma is more commonly used, we use the scaled inverse chi-squared for the sake of convenience. The prior for  $\sigma^2$  is as follows:

$$p(\sigma^2 \mid \nu_0, \sigma_0^2) = \frac{(\sigma_0^2 \frac{\nu_0}{2})^{\nu_0/2}}{\Gamma(\frac{\nu_0}{2})} \frac{\exp\left[\frac{-\nu_0 \sigma_0^2}{2\sigma^2}\right]}{(\sigma^2)^{1+\frac{\nu_0}{2}}} \propto \frac{\exp\left[\frac{-\nu_0 \sigma_0^2}{2\sigma^2}\right]}{(\sigma^2)^{1+\frac{\nu_0}{2}}}$$

The likelihood function from above, written in terms of the variance, is:

$$\begin{aligned}p(\mathbf{X} \mid \mu, \sigma^2) &= \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2\right] \\ &= \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\frac{S}{2\sigma^2}\right]\end{aligned}$$

where

$$S = \sum_{i=1}^n (x_i - \mu)^2.$$

Then:

$$\begin{aligned}
p(\sigma^2 \mid \mathbf{X}) &\propto p(\mathbf{X} \mid \sigma^2)p(\sigma^2) \\
&= \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\frac{S}{2\sigma^2}\right] \frac{(\sigma_0^2 \frac{\nu_0}{2})^{\frac{\nu_0}{2}}}{\Gamma(\frac{\nu_0}{2})} \frac{\exp\left[\frac{-\nu_0\sigma_0^2}{2\sigma^2}\right]}{(\sigma^2)^{1+\frac{\nu_0}{2}}} \\
&\propto \left(\frac{1}{\sigma^2}\right)^{n/2} \frac{1}{(\sigma^2)^{1+\frac{\nu_0}{2}}} \exp\left[-\frac{S}{2\sigma^2} + \frac{-\nu_0\sigma_0^2}{2\sigma^2}\right] \\
&= \frac{1}{(\sigma^2)^{1+\frac{\nu_0+n}{2}}} \exp\left[-\frac{\nu_0\sigma_0^2 + S}{2\sigma^2}\right]
\end{aligned}$$

The above is also a scaled inverse chi-squared distribution where

$$\begin{aligned}
\nu'_0 &= \nu_0 + n \\
\nu'_0\sigma_0'^2 &= \nu_0\sigma_0^2 + \sum_{i=1}^n (x_i - \mu)^2
\end{aligned}$$

or equivalently

$$\begin{aligned}
\nu'_0 &= \nu_0 + n \\
\sigma_0'^2 &= \frac{\nu_0\sigma_0^2 + \sum_{i=1}^n (x_i - \mu)^2}{\nu_0 + n}
\end{aligned}$$

Reparameterizing in terms of an inverse gamma distribution, the result is:

$$\begin{aligned}
\alpha' &= \alpha + \frac{n}{2} \\
\beta' &= \beta + \frac{\sum_{i=1}^n (x_i - \mu)^2}{2}
\end{aligned}$$

### With unknown mean and unknown variance

For a set of i.i.d. normally distributed data points  $\mathbf{X}$  of size  $n$  where each individual point  $x$  follows  $x \sim \mathcal{N}(\mu, \sigma^2)$  with unknown mean  $\mu$  and unknown variance  $\sigma^2$ , a combined (multivariate) conjugate prior is placed over the mean and variance, consisting of a normal-inverse-gamma distribution. Logically, this originates as follows:

1. From the analysis of the case with unknown mean but known variance, we see that the update equations involve sufficient statistics computed from the data consisting of the mean of the data points and the total variance of the data points, computed in turn from the known variance divided by the number of data points.
2. From the analysis of the case with unknown variance but known mean, we see that the update equations involve sufficient statistics over the data consisting of the number of data points and sum of squared deviations.
3. Keep in mind that the posterior update values serve as the prior distribution when further data is handled. Thus, we should logically think of our priors in terms of the sufficient statistics just described, with the same semantics kept in mind as much as possible.
4. To handle the case where both mean and variance are unknown, we could place independent priors over the mean and variance, with fixed estimates of the average mean, total variance, number of data points used to compute the variance prior, and sum of squared deviations. Note however that in reality, the total variance of the mean depends on the unknown variance, and the sum of squared deviations that goes into the variance prior (appears to) depend on the unknown mean. In practice, the latter dependence is relatively unimportant: Shifting the actual mean shifts the generated points by an equal amount, and on average the squared deviations will remain the same. This is not the case, however, with the total variance of the mean: As the unknown variance

increases, the total variance of the mean will increase proportionately, and we would like to capture this dependence.

5. This suggests that we create a *conditional prior* of the mean on the unknown variance, with a hyperparameter specifying the mean of the pseudo-observations associated with the prior, and another parameter specifying the number of pseudo-observations. This number serves as a scaling parameter on the variance, making it possible to control the overall variance of the mean relative to the actual variance parameter. The prior for the variance also has two hyperparameters, one specifying the sum of squared deviations of the pseudo-observations associated with the prior, and another specifying once again the number of pseudo-observations. Note that each of the priors has a hyperparameter specifying the number of pseudo-observations, and in each case this controls the relative variance of that prior. These are given as two separate hyperparameters so that the variance (aka the confidence) of the two priors can be controlled separately.
6. This leads immediately to the normal-inverse-gamma distribution, which is the product of the two distributions just defined, with conjugate priors used (an inverse gamma distribution over the variance, and a normal distribution over the mean, *conditional* on the variance) and with the same four parameters just defined.

The priors are normally defined as follows:

$$p(\mu \mid \sigma^2; \mu_0, n_0) \sim \mathcal{N}(\mu_0, \sigma^2/n_0)$$

$$p(\sigma^2; \nu_0, \sigma_0^2) \sim I\chi^2(\nu_0, \sigma_0^2) = IG(\nu_0/2, \nu_0\sigma_0^2/2)$$

The update equations can be derived, and look as follows:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\mu'_0 = \frac{n_0\mu_0 + n\bar{x}}{n_0 + n}$$

$$n'_0 = n_0 + n$$

$$\nu'_0 = \nu_0 + n$$

$$\nu'_0\sigma_0'^2 = \nu_0\sigma_0^2 + \sum_{i=1}^n (x_i - \bar{x})^2 + \frac{n_0n}{n_0 + n} (\mu_0 - \bar{x})^2$$

The respective numbers of pseudo-observations add the number of actual observations to them. The new mean hyperparameter is once again a weighted average, this time weighted by the relative numbers of observations. Finally, the update for  $\nu'_0\sigma_0'^2$  is similar to the case with known mean, but in this case the sum of squared deviations is taken with respect to the observed data mean rather than the true mean, and as a result a new "interaction term" needs to be added to take care of the additional error source stemming from the deviation between prior and data mean.

### [Proof]

The prior distributions are

$$p(\mu \mid \sigma^2; \mu_0, n_0) \sim \mathcal{N}(\mu_0, \sigma^2/n_0) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{n_0}}} \exp\left(-\frac{n_0}{2\sigma^2}(\mu - \mu_0)^2\right)$$

$$\propto (\sigma^2)^{-1/2} \exp\left(-\frac{n_0}{2\sigma^2}(\mu - \mu_0)^2\right)$$

$$p(\sigma^2; \nu_0, \sigma_0^2) \sim I\chi^2(\nu_0, \sigma_0^2) = IG(\nu_0/2, \nu_0\sigma_0^2/2)$$

$$= \frac{(\sigma_0^2\nu_0/2)^{\nu_0/2}}{\Gamma(\nu_0/2)} \frac{\exp\left[\frac{-\nu_0\sigma_0^2}{2\sigma^2}\right]}{(\sigma^2)^{1+\nu_0/2}}$$

$$\propto (\sigma^2)^{-(1+\nu_0/2)} \exp\left[\frac{-\nu_0\sigma_0^2}{2\sigma^2}\right].$$

Therefore, the joint prior is

$$\begin{aligned} p(\mu, \sigma^2; \mu_0, n_0, \nu_0, \sigma_0^2) &= p(\mu \mid \sigma^2; \mu_0, n_0) p(\sigma^2; \nu_0, \sigma_0^2) \\ &\propto (\sigma^2)^{-(\nu_0+3)/2} \exp\left[-\frac{1}{2\sigma^2} (\nu_0\sigma_0^2 + n_0(\mu - \mu_0)^2)\right]. \end{aligned}$$

The likelihood function from the section above with known variance is:

$$p(\mathbf{X} \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\frac{1}{2\sigma^2} \left(\sum_{i=1}^n (x_i - \mu)^2\right)\right]$$

Writing it in terms of variance rather than precision, we get:

$$\begin{aligned} p(\mathbf{X} \mid \mu, \sigma^2) &= \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\frac{1}{2\sigma^2} \left(\sum_{i=1}^n (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2\right)\right] \\ &\propto \sigma^{2-n/2} \exp\left[-\frac{1}{2\sigma^2} (S + n(\bar{x} - \mu)^2)\right] \end{aligned}$$

where  $S = \sum_{i=1}^n (x_i - \bar{x})^2$ .

Therefore, the posterior is (dropping the hyperparameters as conditioning factors):

$$\begin{aligned} p(\mu, \sigma^2 \mid \mathbf{X}) &\propto p(\mu, \sigma^2) p(\mathbf{X} \mid \mu, \sigma^2) \\ &\propto (\sigma^2)^{-(\nu_0+3)/2} \exp\left[-\frac{1}{2\sigma^2} (\nu_0\sigma_0^2 + n_0(\mu - \mu_0)^2)\right] \sigma^{2-n/2} \exp\left[-\frac{1}{2\sigma^2} (S + n(\bar{x} - \mu)^2)\right] \\ &= (\sigma^2)^{-(\nu_0+n+3)/2} \exp\left[-\frac{1}{2\sigma^2} (\nu_0\sigma_0^2 + S + n_0(\mu - \mu_0)^2 + n(\bar{x} - \mu)^2)\right] \\ &= (\sigma^2)^{-(\nu_0+n+3)/2} \exp\left[-\frac{1}{2\sigma^2} \left(\nu_0\sigma_0^2 + S + \frac{n_0n}{n_0+n}(\mu_0 - \bar{x})^2 + (n_0+n)\left(\mu - \frac{n_0\mu_0 + n\bar{x}}{n_0+n}\right)^2\right)\right] \\ &\propto (\sigma^2)^{-1/2} \exp\left[-\frac{n_0+n}{2\sigma^2} \left(\mu - \frac{n_0\mu_0 + n\bar{x}}{n_0+n}\right)^2\right] \\ &\quad \times (\sigma^2)^{-(\nu_0/2+n/2+1)} \exp\left[-\frac{1}{2\sigma^2} \left(\nu_0\sigma_0^2 + S + \frac{n_0n}{n_0+n}(\mu_0 - \bar{x})^2\right)\right] \\ &= \mathcal{N}_{\mu|\sigma^2} \left( \frac{n_0\mu_0 + n\bar{x}}{n_0+n}, \frac{\sigma^2}{n_0+n} \right) \cdot \text{IG}_{\sigma^2} \left( \frac{1}{2}(\nu_0+n), \frac{1}{2} \left( \nu_0\sigma_0^2 + S + \frac{n_0n}{n_0+n}(\mu_0 - \bar{x})^2 \right) \right). \end{aligned}$$

In other words, the posterior distribution has the form of a product of a normal distribution over  $p(\mu \mid \sigma^2)$  times an inverse gamma distribution over  $p(\sigma^2)$ , with parameters that are the same as the update equations above.

## Occurrence and applications

The occurrence of normal distribution in practical problems can be loosely classified into four categories:

1. Exactly normal distributions;
2. Approximately normal laws, for example when such approximation is justified by the central limit theorem; and
3. Distributions modeled as normal – the normal distribution being the distribution with maximum entropy for a given mean and variance.

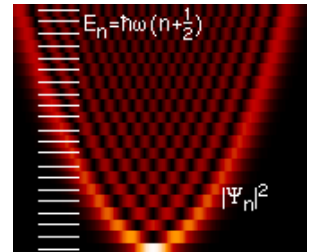
4. Regression problems – the normal distribution being found after systematic effects have been modeled sufficiently well.

## Exact normality

Certain quantities in physics are distributed normally, as was first demonstrated by James Clerk Maxwell. Examples of such quantities are:

- Probability density function of a ground state in a quantum harmonic oscillator.
- The position of a particle that experiences diffusion. If initially the particle is located at a specific point (that is its probability distribution is the Dirac delta function), then after time  $t$  its location is described by a normal distribution with variance  $t$ , which satisfies

the diffusion equation  $\frac{\partial}{\partial t} f(x, t) = \frac{1}{2} \frac{\partial^2}{\partial x^2} f(x, t)$ . If the initial location is given by a certain density function  $g(x)$ , then the density at time  $t$  is the convolution of  $g$  and the normal PDF.



The ground state of a quantum harmonic oscillator has the Gaussian distribution.

## Approximate normality

*Approximately* normal distributions occur in many situations, as explained by the central limit theorem. When the outcome is produced by many small effects acting *additively and independently*, its distribution will be close to normal. The normal approximation will not be valid if the effects act multiplicatively (instead of additively), or if there is a single external influence that has a considerably larger magnitude than the rest of the effects.

- In counting problems, where the central limit theorem includes a discrete-to-continuum approximation and where infinitely divisible and decomposable distributions are involved, such as
  - Binomial random variables, associated with binary response variables;
  - Poisson random variables, associated with rare events;
- Thermal radiation has a Bose–Einstein distribution on very short time scales, and a normal distribution on longer timescales due to the central limit theorem.

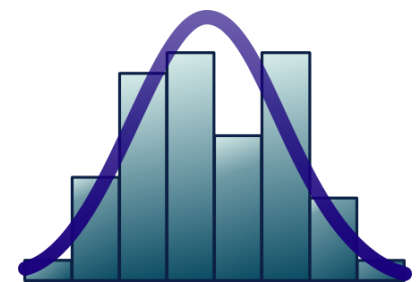
## Assumed normality

I can only recognize the occurrence of the normal curve – the Laplacian curve of errors – as a very abnormal phenomenon. It is roughly approximated to in certain distributions; for this reason, and on account for its beautiful simplicity, we may, perhaps, use it as a first approximation, particularly in theoretical investigations.

— Pearson (1901)

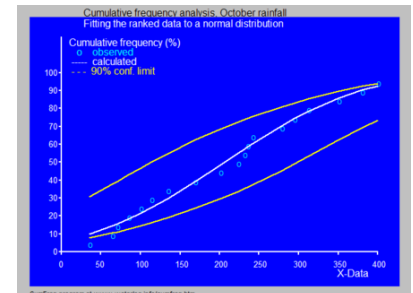
There are statistical methods to empirically test that assumption, see the above Normality tests section.

- In biology, the *logarithm* of various variables tend to have a normal distribution, that is, they tend to have a log-normal distribution (after separation on male/female subpopulations), with examples including:
  - Measures of size of living tissue (length, height, skin area, weight);<sup>[52]</sup>
  - The *length of inert* appendages (hair, claws, nails, teeth) of biological specimens, *in the direction of growth*; presumably the thickness of tree bark also falls under this category;
  - Certain physiological measurements, such as blood pressure of adult humans.



Histogram of sepal widths for *Iris versicolor* from Fisher's Iris flower data set, with superimposed best-fitting normal distribution.

- In finance, in particular the Black–Scholes model, changes in the *logarithm* of exchange rates, price indices, and stock market indices are assumed normal (these variables behave like compound interest, not like simple interest, and so are multiplicative). Some mathematicians such as Benoit Mandelbrot have argued that log-Levy distributions, which possesses heavy tails would be a more appropriate model, in particular for the analysis for stock market crashes. The use of the assumption of normal distribution occurring in financial models has also been criticized by Nassim Nicholas Taleb in his works.
- Measurement errors in physical experiments are often modeled by a normal distribution. This use of a normal distribution does not imply that one is assuming the measurement errors are normally distributed, rather using the normal distribution produces the most conservative predictions possible given only knowledge about the mean and variance of the errors.<sup>[53]</sup>
- In standardized testing, results can be made to have a normal distribution by either selecting the number and difficulty of questions (as in the IQ test) or transforming the raw test scores into "output" scores by fitting them to the normal distribution. For example, the SAT's traditional range of 200–800 is based on a normal distribution with a mean of 500 and a standard deviation of 100.
- Many scores are derived from the normal distribution, including percentile ranks ("percentiles" or "quantiles"), normal curve equivalents, stanines, z-scores, and T-scores. Additionally, some behavioral statistical procedures assume that scores are normally distributed; for example, t-tests and ANOVAs. Bell curve grading assigns relative grades based on a normal distribution of scores.
- In hydrology the distribution of long duration river discharge or rainfall, e.g. monthly and yearly totals, is often thought to be practically normal according to the central limit theorem.<sup>[54]</sup> The blue picture, made with CumFreq, illustrates an example of fitting the normal distribution to ranked October rainfalls showing the 90% confidence belt based on the binomial distribution. The rainfall data are represented by plotting positions as part of the cumulative frequency analysis.



Fitted cumulative normal distribution to October rainfalls, see [distribution fitting](#)

## Produced normality

In regression analysis, lack of normality in residuals simply indicates that the model postulated is inadequate in accounting for the tendency in the data and needs to be augmented; in other words, normality in residuals can always be achieved given a properly constructed model.

## Computational methods

### Generating values from normal distribution

In computer simulations, especially in applications of the Monte-Carlo method, it is often desirable to generate values that are normally distributed. The algorithms listed below all generate the standard normal deviates, since a  $N(\mu, \sigma^2)$  can be generated as  $X = \mu + \sigma Z$ , where  $Z$  is standard normal. All these algorithms rely on the availability of a random number generator  $U$  capable of producing uniform random variates.

- The most straightforward method is based on the probability integral transform property: if  $U$  is distributed uniformly on  $(0,1)$ , then  $\Phi^{-1}(U)$  will have the standard normal distribution. The drawback of this method is that it relies on calculation of the probit function  $\Phi^{-1}$ , which cannot be done analytically. Some approximate methods are described in Hart (1968) and in the erf article. Wichura gives a fast algorithm for computing this function to 16 decimal places,<sup>[55]</sup> which is used by R to compute random variates of the normal distribution.
- An easy to program approximate approach, that relies on the central limit theorem, is as follows: generate 12 uniform  $U(0,1)$  deviates, add them all up, and subtract 6 – the resulting random variable will have approximately standard normal distribution. In truth, the distribution will be Irwin–Hall, which is a 12-section eleventh-order polynomial approximation to the normal distribution. This random deviate will have a limited range of  $(-6, 6)$ .<sup>[56]</sup>
- The Box–Muller method uses two independent random numbers  $U$  and  $V$  distributed uniformly on  $(0,1)$ . Then the two random variables  $X$  and  $Y$

$$X = \sqrt{-2 \ln U} \cos(2\pi V), \quad Y = \sqrt{-2 \ln U} \sin(2\pi V).$$

will both have the standard normal distribution, and will be independent. This formulation arises because for a bivariate normal random vector  $(X, Y)$  the squared norm  $X^2 + Y^2$  will have the chi-squared distribution with two degrees of freedom, which is an easily generated exponential random variable corresponding to the quantity  $-2\ln(U)$  in these equations; and the angle is distributed uniformly around the circle, chosen by the random variable  $V$ .

- The Marsaglia polar method is a modification of the Box–Muller method which does not require computation of the sine and cosine functions. In this method,  $U$  and  $V$  are drawn from the uniform  $(-1, 1)$  distribution, and then  $S = U^2 + V^2$  is computed. If  $S$  is greater or equal to 1, then the method starts over, otherwise the two quantities

$$X = U \sqrt{\frac{-2 \ln S}{S}}, \quad Y = V \sqrt{\frac{-2 \ln S}{S}}$$

are returned. Again,  $X$  and  $Y$  are independent, standard normal random variables.

- The Ratio method<sup>[57]</sup> is a rejection method. The algorithm proceeds as follows:
  - Generate two independent uniform deviates  $U$  and  $V$ ;
  - Compute  $X = \sqrt{8/e} (V - 0.5)/U$ ;
  - Optional: if  $X^2 \leq 5 - 4e^{1/4}U$  then accept  $X$  and terminate algorithm;
  - Optional: if  $X^2 \geq 4e^{-1.35}/U + 1.4$  then reject  $X$  and start over from step 1;
  - If  $X^2 \leq -4 \ln U$  then accept  $X$ , otherwise start over the algorithm.

The two optional steps allow the evaluation of the logarithm in the last step to be avoided in most cases. These steps can be greatly improved<sup>[58]</sup> so that the logarithm is rarely evaluated.

- The ziggurat algorithm<sup>[59]</sup> is faster than the Box–Muller transform and still exact. In about 97% of all cases it uses only two random numbers, one random integer and one random uniform, one multiplication and an if-test. Only in 3% of the cases, where the combination of those two falls outside the "core of the ziggurat" (a kind of rejection sampling using logarithms), do exponentials and more uniform random numbers have to be employed.
- Integer arithmetic can be used to sample from the standard normal distribution.<sup>[60]</sup> This method is exact in the sense that it satisfies the conditions of *ideal approximation*;<sup>[61]</sup> i.e., it is equivalent to sampling a real number from the standard normal distribution and rounding this to the nearest representable floating point number.
- There is also some investigation<sup>[62]</sup> into the connection between the fast Hadamard transform and the normal distribution, since the transform employs just addition and subtraction and by the central limit theorem random numbers from almost any distribution will be transformed into the normal distribution. In this regard a series of Hadamard transforms can be combined with random permutations to turn arbitrary data sets into a normally distributed data.



The bean machine, a device invented by Francis Galton, can be called the first generator of normal random variables. This machine consists of a vertical board with interleaved rows of pins. Small balls are dropped from the top and then bounce randomly left or right as they hit the pins. The balls are collected into bins at the bottom and settle down into a pattern resembling the Gaussian curve.

## Numerical approximations for the normal CDF

The standard normal CDF is widely used in scientific and statistical computing.

The values  $\Phi(x)$  may be approximated very accurately by a variety of methods, such as numerical integration, Taylor series, asymptotic series and continued fractions. Different approximations are used depending on the desired level of accuracy.

- Zelen & Severo (1964) give the approximation for  $\Phi(x)$  for  $x > 0$  with the absolute error  $|\varepsilon(x)| < 7.5 \cdot 10^{-8}$  (algorithm 26.2.17 ([http://www.math.sfu.ca/~cbm/aands/page\\_932.htm](http://www.math.sfu.ca/~cbm/aands/page_932.htm))):

$$\Phi(x) = 1 - \varphi(x) (b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5) + \varepsilon(x), \quad t = \frac{1}{1 + b_0 x},$$

where  $\phi(x)$  is the standard normal PDF, and  $b_0 = 0.2316419$ ,  $b_1 = 0.319381530$ ,  $b_2 = -0.356563782$ ,  $b_3 = 1.781477937$ ,  $b_4 = -1.821255978$ ,  $b_5 = 1.330274429$ .

- Hart (1968) lists some dozens of approximations – by means of rational functions, with or without exponentials – for the  $\operatorname{erfc}()$  function. His algorithms vary in the degree of complexity and the resulting precision, with maximum absolute precision of 24 digits. An algorithm by West (2009) combines Hart's algorithm 5666 with a continued fraction approximation in the tail to provide a fast computation algorithm with a 16-digit precision.
- Cody (1969) after recalling Hart68 solution is not suited for erf, gives a solution for both erf and erfc, with maximal relative error bound, via Rational Chebyshev Approximation.
- Marsaglia (2004) suggested a simple algorithm<sup>[note 2]</sup> based on the Taylor series expansion

$$\Phi(x) = \frac{1}{2} + \varphi(x) \left( x + \frac{x^3}{3} + \frac{x^5}{3 \cdot 5} + \frac{x^7}{3 \cdot 5 \cdot 7} + \frac{x^9}{3 \cdot 5 \cdot 7 \cdot 9} + \dots \right)$$

for calculating  $\Phi(x)$  with arbitrary precision. The drawback of this algorithm is comparatively slow calculation time (for example it takes over 300 iterations to calculate the function with 16 digits of precision when  $x = 10$ ).

- The GNU Scientific Library calculates values of the standard normal CDF using Hart's algorithms and approximations with Chebyshev polynomials.

Shore (1982) introduced simple approximations that may be incorporated in stochastic optimization models of engineering and operations research, like reliability engineering and inventory analysis. Denoting  $p = \Phi(z)$ , the simplest approximation for the quantile function is:

$$z = \Phi^{-1}(p) = 5.5556 \left[ 1 - \left( \frac{1-p}{p} \right)^{0.1186} \right], \quad p \geq 1/2$$

This approximation delivers for  $z$  a maximum absolute error of 0.026 (for  $0.5 \leq p \leq 0.9999$ , corresponding to  $0 \leq z \leq 3.719$ ). For  $p < 1/2$  replace  $p$  by  $1 - p$  and change sign. Another approximation, somewhat less accurate, is the single-parameter approximation:

$$z = -0.4115 \left\{ \frac{1-p}{p} + \log \left[ \frac{1-p}{p} \right] - 1 \right\}, \quad p \geq 1/2$$

The latter had served to derive a simple approximation for the loss integral of the normal distribution, defined by

$$L(z) = \int_z^\infty (u - z) \varphi(u) du = \int_z^\infty [1 - \Phi(u)] du$$

$$L(z) \approx \begin{cases} 0.4115 \left( \frac{p}{1-p} \right) - z, & p < 1/2, \\ 0.4115 \left( \frac{1-p}{p} \right), & p \geq 1/2. \end{cases}$$

or, equivalently,

$$L(z) \approx \begin{cases} 0.4115 \left\{ 1 - \log \left[ \frac{p}{1-p} \right] \right\}, & p < 1/2, \\ 0.4115 \frac{1-p}{p}, & p \geq 1/2. \end{cases}$$

This approximation is particularly accurate for the right far-tail (maximum error of  $10^{-3}$  for  $z \geq 1.4$ ). Highly accurate approximations for the CDF, based on Response Modeling Methodology (RMM, Shore, 2011, 2012), are shown in Shore (2005).

Some more approximations can be found at: Error function#Approximation with elementary functions. In particular, small *relative* error on the whole domain for the CDF  $\Phi$  and the quantile function  $\Phi^{-1}$  as well, is achieved via an explicitly invertible formula by Sergei Winitzki in 2008.

## History

### Development

Some authors<sup>[63][64]</sup> attribute the credit for the discovery of the normal distribution to de Moivre, who in 1738<sup>[note 3]</sup> published in the second edition of his "*The Doctrine of Chances*" the study of the coefficients in the binomial expansion of  $(a + b)^n$ . De Moivre proved that the middle term in this expansion has the approximate magnitude of  $2/\sqrt{2\pi n}$ , and that "If  $m$  or  $1/2n$  be a Quantity infinitely great, then the Logarithm of the Ratio, which a Term distant from the middle by the Interval  $\ell$ , has to the middle Term, is  $-\frac{2\ell\ell}{n}$ ."<sup>[65]</sup> Although this theorem can be interpreted as the first obscure expression for the normal probability law, Stigler points out that de Moivre himself did not interpret his results as anything more than the approximate rule for the binomial coefficients, and in particular de Moivre lacked the concept of the probability density function.<sup>[66]</sup>



Carl Friedrich Gauss discovered the normal distribution in 1809 as a way to rationalize the method of least squares.

In 1809 Gauss published his monograph "*Theoria motus corporum coelestium in sectionibus conicis solem ambientium*" where among other things he introduces several important statistical concepts, such as the method of least squares, the method of maximum likelihood, and the *normal distribution*. Gauss used  $M, M', M'', \dots$  to denote the measurements of some unknown quantity  $V$ , and sought the "most probable" estimator of that quantity: the one that maximizes the probability  $\varphi(M - V) \cdot \varphi(M' - V) \cdot \varphi(M'' - V) \cdot \dots$  of obtaining the observed experimental results. In his notation  $\varphi\Delta$  is the probability law of the measurement errors of magnitude  $\Delta$ . Not knowing what the function  $\varphi$  is, Gauss requires that his method should reduce to the well-known answer: the arithmetic mean of the measured values.<sup>[note 4]</sup> Starting from these principles, Gauss demonstrates that the only law that rationalizes the choice of arithmetic mean as an estimator of the location parameter, is the normal law of errors:<sup>[67]</sup>

$$\varphi\Delta = \frac{h}{\sqrt{\pi}} e^{-h^2\Delta^2},$$

where  $h$  is "the measure of the precision of the observations". Using this normal law as a generic model for errors in the experiments, Gauss formulates what is now known as the non-linear weighted least squares (NWLS) method.<sup>[68]</sup>

Although Gauss was the first to suggest the normal distribution law, Laplace made significant contributions.<sup>[note 5]</sup> It was Laplace who first posed the problem of aggregating several observations in 1774,<sup>[69]</sup> although his own solution led to the Laplacian distribution. It was Laplace who first calculated the value of the integral  $\int e^{-t^2} dt = \sqrt{\pi}$  in 1782, providing the normalization constant for the normal distribution.<sup>[70]</sup> Finally, it was Laplace who in 1810 proved and presented to the Academy the fundamental central limit theorem, which emphasized the theoretical importance of the normal distribution.<sup>[71]</sup>

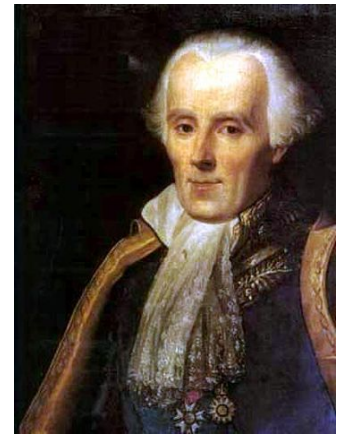
It is of interest to note that in 1809 an Irish mathematician Adrain published two derivations of the normal probability law, simultaneously and independently from Gauss.<sup>[72]</sup> His works remained largely unnoticed by the scientific community, until in 1871 they were "rediscovered" by Abbe.<sup>[73]</sup>

In the middle of the 19th century Maxwell demonstrated that the normal distribution is not just a convenient mathematical tool, but may also occur in natural phenomena:<sup>[74]</sup> "The number of particles whose velocity, resolved in a certain direction, lies between  $x$  and  $x + dx$  is

$$N \frac{1}{\alpha \sqrt{\pi}} e^{-\frac{x^2}{\alpha^2}} dx$$

## Naming

Since its introduction, the normal distribution has been known by many different names: the law of error, the law of facility of errors, Laplace's second law, Gaussian law, etc. Gauss himself apparently coined the term with reference to the "normal equations" involved in its applications, with normal having its technical meaning of orthogonal rather than "usual".<sup>[75]</sup> However, by the end of the 19th century some authors<sup>[note 6]</sup> had started using the name *normal distribution*, where the word "normal" was used as an adjective – the term now being seen as a reflection of the fact that this distribution was seen as typical, common – and thus "normal". Peirce (one of those authors) once defined "normal" thus: "...the 'normal' is not the average (or any other kind of mean) of what actually occurs, but of what *would*, in the long run, occur under certain circumstances."<sup>[76]</sup> Around the turn of the 20th century Pearson popularized the term *normal* as a designation for this distribution.<sup>[77]</sup>



Pierre-Simon Laplace proved the central limit theorem in 1810, consolidating the importance of the normal distribution in statistics.

Many years ago I called the Laplace–Gaussian curve the *normal* curve, which name, while it avoids an international question of priority, has the disadvantage of leading people to believe that all other distributions of frequency are in one sense or another 'abnormal'.

— Pearson (1920)

Also, it was Pearson who first wrote the distribution in terms of the standard deviation  $\sigma$  as in modern notation. Soon after this, in year 1915, Fisher added the location parameter to the formula for normal distribution, expressing it in the way it is written nowadays:

$$df = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-(x-m)^2/(2\sigma^2)} dx$$

The term "standard normal", which denotes the normal distribution with zero mean and unit variance came into general use around the 1950s, appearing in the popular textbooks by P.G. Hoel (1947) "*Introduction to mathematical statistics*" and A.M. Mood (1950) "*Introduction to the theory of statistics*".<sup>[78]</sup>

## See also

- Bates distribution — similar to the Irwin–Hall distribution, but rescaled back into the 0 to 1 range
- Behrens–Fisher problem — the long-standing problem of testing whether two normal samples with different variances have same means;
- Bhattacharyya distance – method used to separate mixtures of normal distributions
- Erdős–Kac theorem—on the occurrence of the normal distribution in number theory
- Gaussian blur—convolution, which uses the normal distribution as a kernel
- Normally distributed and uncorrelated does not imply independent
- Reciprocal normal distribution
- Ratio normal distribution
- Standard normal table
- Stein's lemma

- Sub-Gaussian distribution
- Sum of normally distributed random variables
- Tweedie distribution — The normal distribution is a member of the family of Tweedie exponential dispersion models
- Wrapped normal distribution — the Normal distribution applied to a circular domain
- Z-test— using the normal distribution

## Notes

---

1. For the proof see Gaussian integral.
2. For example, this algorithm is given in the article Bc programming language.
3. De Moivre first published his findings in 1733, in a pamphlet "Approximatio ad Summam Terminorum Binomii  $(a + b)^n$  in Seriem Expansi" that was designated for private circulation only. But it was not until the year 1738 that he made his results publicly available. The original pamphlet was reprinted several times, see for example Walker (1985).
4. "It has been customary certainly to regard as an axiom the hypothesis that if any quantity has been determined by several direct observations, made under the same circumstances and with equal care, the arithmetical mean of the observed values affords the most probable value, if not rigorously, yet very nearly at least, so that it is always most safe to adhere to it." — Gauss (1809, section 177)
5. "My custom of terming the curve the Gauss–Laplacian or *normal* curve saves us from proportioning the merit of discovery between the two great astronomer mathematicians." quote from Pearson (1905, p. 189)
6. Besides those specifically referenced here, such use is encountered in the works of Peirce, Galton (Galton (1889, chapter V)) and Lexis (Lexis (1878), Rohrbasser & Véron (2003)) c. 1875.

## References

---

### Citations

1. "List of Probability and Statistics Symbols" (<https://mathvault.ca/hub/higher-math/math-symbols/probability-statistics-symbols/>). *Math Vault*. April 26, 2020. Retrieved August 15, 2020.
2. Weisstein, Eric W. "Normal Distribution" (<https://mathworld.wolfram.com/NormalDistribution.html>). *mathworld.wolfram.com*. Retrieved August 15, 2020.
3. *Normal Distribution* ([http://www.encyclopedia.com/topic/Normal\\_Distribution.aspx#3](http://www.encyclopedia.com/topic/Normal_Distribution.aspx#3)), Gale Encyclopedia of Psychology
4. Casella & Berger (2001, p. 102)
5. Lyon, A. (2014). *Why are Normal Distributions Normal?* ([https://aidanlyon.com/normal\\_distributions.pdf](https://aidanlyon.com/normal_distributions.pdf)), The British Journal for the Philosophy of Science.
6. "Normal Distribution" (<https://www.mathsisfun.com/data/standard-normal-distribution.html>). *www.mathsisfun.com*. Retrieved August 15, 2020.
7. Stigler (1982)
8. Halperin, Hartley & Hoel (1965, item 7)
9. McPherson (1990, p. 110)
10. Bernardo & Smith (2000, p. 121)
11. Scott, Clayton; Nowak, Robert (August 7, 2003). "The Q-function" (<http://cnx.org/content/m11537/1.2/>). *Connexions*.
12. Barak, Ohad (April 6, 2006). "Q Function and Error Function" (<https://web.archive.org/web/20090325160012/http://www.eng.tau.ac.il/~jo/academic/Q.pdf>) (PDF). Tel Aviv University. Archived from the original (<http://www.eng.tau.ac.il/~jo/academic/Q.pdf>) (PDF) on March 25, 2009.
13. Weisstein, Eric W. "Normal Distribution Function" (<https://mathworld.wolfram.com/NormalDistributionFunction.html>). *MathWorld*.

14. Abramowitz, Milton; Stegun, Irene Ann, eds. (1983) [June 1964]. "Chapter 26, eqn 26.2.12" ([http://www.math.sfu.ca/~cbm/aands/page\\_932.htm](http://www.math.sfu.ca/~cbm/aands/page_932.htm)). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Applied Mathematics Series. **55** (Ninth reprint with additional corrections of tenth original printing with corrections (December 1972); first ed.). Washington D.C.; New York: United States Department of Commerce, National Bureau of Standards; Dover Publications. p. 932. ISBN 978-0-486-61272-0. LCCN 64-60036 (<https://lccn.loc.gov/64-60036>). MR 0167642 (<https://www.ams.org/mathscinet-getitem?mr=0167642>). LCCN 65-12253 (<https://lccn.loc.gov/65012253>).
15. "Wolfram|Alpha: Computational Knowledge Engine" ([http://www.wolframalpha.com/input/?i=Table%5B{N\(Erf\(n/Sqrt\(2\)\),+12\),+N\(1-Erf\(n/Sqrt\(2\)\),+12\),+N\(1/\(1-Erf\(n/Sqrt\(2\)\)\),+12\)},+{n,1,6}%5D](http://www.wolframalpha.com/input/?i=Table%5B{N(Erf(n/Sqrt(2)),+12),+N(1-Erf(n/Sqrt(2)),+12),+N(1/(1-Erf(n/Sqrt(2))),+12)},+{n,1,6}%5D)). *Wolframalpha.com*. Retrieved March 3, 2017.
16. "Wolfram|Alpha: Computational Knowledge Engine" ([http://www.wolframalpha.com/input/?i=Table%5BSqrt%282%29\\*InverseErf%28x%29%2C+{x%2C+N%288%2F10%2C+9%2F10%2C+19%2F20%2C+49%2F50%2C+99%2F100%2C+995%2F1000%2C+998%2F1000}%2C+13%29}%5D](http://www.wolframalpha.com/input/?i=Table%5BSqrt%282%29*InverseErf%28x%29%2C+{x%2C+N%288%2F10%2C+9%2F10%2C+19%2F20%2C+49%2F50%2C+99%2F100%2C+995%2F1000%2C+998%2F1000}%2C+13%29}%5D)). *Wolframalpha.com*.
17. "Wolfram|Alpha: Computational Knowledge Engine" ([http://www.wolframalpha.com/input/?i=Table%5B%7BN\(1-10%5E\(-x\),9\),N\(Sqrt\(2\)\\*InverseErf\(1-10%5E\(-x\)\),13\)%7D,%7Bx,3,9%7D%5D](http://www.wolframalpha.com/input/?i=Table%5B%7BN(1-10%5E(-x),9),N(Sqrt(2)*InverseErf(1-10%5E(-x)),13)%7D,%7Bx,3,9%7D%5D)). *Wolframalpha.com*. Retrieved March 3, 2017.
18. Cover, Thomas M.; Thomas, Joy A. (2006). *Elements of Information Theory* (<https://archive.org/details/elementsinfoformat00cove>). John Wiley and Sons. p. 254 (<https://archive.org/details/elementsinfoformat00cove/page/n279>).
19. Park, Sung Y.; Bera, Anil K. (2009). "Maximum Entropy Autoregressive Conditional Heteroskedasticity Model" (<http://www.wise.xmu.edu.cn/Master/Download/.%5C..%5CUploadFiles%5Cpaper-masterdownload%5C2009519932327055475115776.pdf>) (PDF). *Journal of Econometrics*. **150** (2): 219–230. CiteSeerX 10.1.1.511.9750 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.511.9750>). doi:10.1016/j.jeconom.2008.12.014 (<https://doi.org/10.1016/j.jeconom.2008.12.014>). Retrieved June 2, 2011.
20. Geary RC(1936) The distribution of the "Student's" ratio for the non-normal samples". Supplement to the Journal of the Royal Statistical Society 3 (2): 178–184
21. Lukas E (1942) A characterization of the normal distribution. Annals of Mathematical Statistics 13: 91–93
22. Patel & Read (1996, [2.1.4])
23. Fan (1991, p. 1258)
24. Patel & Read (1996, [2.1.8])
25. Papoulis, Athanasios. *Probability, Random Variables and Stochastic Processes (4th Edition)*. p. 148.
26. Bryc (1995, p. 23)
27. Bryc (1995, p. 24)
28. Cover & Thomas (2006, p. 254)
29. Williams, David (2001). *Weighing the odds : a course in probability and statistics* (<https://archive.org/details/weighingoddscur00will>) (Reprinted. ed.). Cambridge [u.a.]: Cambridge Univ. Press. pp. 197 (<https://archive.org/details/weighingoddscur00will/page/n219>)–199. ISBN 978-0-521-00618-7.
30. Smith, José M. Bernardo; Adrian F. M. (2000). *Bayesian theory* ([https://archive.org/details/bayesiantheory00bern\\_963](https://archive.org/details/bayesiantheory00bern_963)) (Reprint ed.). Chichester [u.a.]: Wiley. pp. 209 ([https://archive.org/details/bayesiantheory00bern\\_963/page/n224](https://archive.org/details/bayesiantheory00bern_963/page/n224)), 366. ISBN 978-0-471-49464-5.
31. O'Hagan, A. (1994) *Kendall's Advanced Theory of statistics, Vol 2B, Bayesian Inference*, Edward Arnold. ISBN 0-340-52922-9 (Section 5.40)
32. Bryc (1995, p. 27)
33. Patel & Read (1996, [2.3.6])
34. Galambos & Simonelli (2004, Theorem 3.5)
35. Bryc (1995, p. 35)
36. Lukacs & King (1954)
37. Quine, M.P. (1993). "On three characterisations of the normal distribution" (<http://www.math.uni.wroc.pl/~pms/publicationsArticle.php?nr=14.2&nrA=8&ppB=257&ppE=263>). *Probability and Mathematical Statistics*. **14** (2): 257–263.
38. UIUC, Lecture 21. *The Multivariate Normal Distribution* (<http://www.math.uiuc.edu/~r-ash/Stat/StatLec21-25.pdf>), 21.6:"Individually Gaussian Versus Jointly Gaussian".
39. Edward L. Melnick and Aaron Tenenbein, "Misspecifications of the Normal Distribution", *The American Statistician*, volume 36, number 4 November 1982, pages 372–373

40. "Kullback Leibler (KL) Distance of Two Normal (Gaussian) Probability Distributions" (<http://www.allisons.org/ll/MM/L/KL/Normal/>). *Allisons.org*. December 5, 2007. Retrieved March 3, 2017.
41. Jordan, Michael I. (February 8, 2010). "Stat260: Bayesian Modeling and Inference: The Conjugate Prior for the Normal Distribution" (<http://www.cs.berkeley.edu/~jordan/courses/260-spring10/lectures/lecture5.pdf>) (PDF).
42. Amari & Nagaoka (2000)
43. "Normal Approximation to Poisson Distribution" ([http://www.stat.ucla.edu/~dinov/courses\\_students.dir/Applelets.dir/NormalApprox2PoissonApplet.html](http://www.stat.ucla.edu/~dinov/courses_students.dir/Applelets.dir/NormalApprox2PoissonApplet.html)). *Stat.ucla.edu*. Retrieved March 3, 2017.
44. Weisstein, Eric W. "Normal Product Distribution" (<http://mathworld.wolfram.com/NormalProductDistribution.html>). *MathWorld*. wolfram.com.
45. Lukacs, Eugene (1942). "A Characterization of the Normal Distribution" (<https://doi.org/10.1214/aoms/1177731647>). *The Annals of Mathematical Statistics*. **13** (1): 91–3. doi:10.1214/aoms/1177731647 (<https://doi.org/10.1214%2Faoms%2F1177731647>). ISSN 0003-4851 (<https://www.worldcat.org/issn/0003-4851>). JSTOR 2236166 (<https://www.jstor.org/stable/2236166>).
46. Basu, D.; Laha, R. G. (1954). "On Some Characterizations of the Normal Distribution". *Sankhyā*. **13** (4): 359–62. ISSN 0036-4452 (<https://www.worldcat.org/issn/0036-4452>). JSTOR 25048183 (<https://www.jstor.org/stable/25048183>).
47. Lehmann, E. L. (1997). *Testing Statistical Hypotheses* (2nd ed.). Springer. p. 199. ISBN 978-0-387-94919-2.
48. John, S (1982). "The three parameter two-piece normal family of distributions and its fitting". *Communications in Statistics - Theory and Methods*. **11** (8): 879–885. doi:10.1080/03610928208828279 (<https://doi.org/10.1080%2F03610928208828279>).
49. Krishnamoorthy (2006, p. 127)
50. Krishnamoorthy (2006, p. 130)
51. Krishnamoorthy (2006, p. 133)
52. Huxley (1932)
53. Jaynes, Edwin T. (2003). *Probability Theory: The Logic of Science* (<https://books.google.com/books?id=tTN4HuUNXjgC&pg=PA592>). Cambridge University Press. pp. 592–593. ISBN 9780521592710.
54. Oosterbaan, Roland J. (1994). "Chapter 6: Frequency and Regression Analysis of Hydrologic Data" (<http://www.waterlog.info/pdf/freqtxt.pdf>) (PDF). In Ritzema, Henk P. (ed.). *Drainage Principles and Applications, Publication 16* (second revised ed.). Wageningen, The Netherlands: International Institute for Land Reclamation and Improvement (ILRI). pp. 175–224. ISBN 978-90-70754-33-4.
55. Wichura, Michael J. (1988). "Algorithm AS241: The Percentage Points of the Normal Distribution". *Applied Statistics*. **37** (3): 477–84. doi:10.2307/2347330 (<https://doi.org/10.2307%2F2347330>). JSTOR 2347330 (<https://www.jstor.org/stable/2347330>).
56. Johnson, Kotz & Balakrishnan (1995, Equation (26.48))
57. Kinderman & Monahan (1977)
58. Leva (1992)
59. Marsaglia & Tsang (2000)
60. Karney (2016)
61. Monahan (1985, section 2)
62. Wallace (1996)
63. Johnson, Kotz & Balakrishnan (1994, p. 85)
64. Le Cam & Lo Yang (2000, p. 74)
65. De Moivre, Abraham (1733), Corollary I – see Walker (1985, p. 77)
66. Stigler (1986, p. 76)
67. Gauss (1809, section 177)
68. Gauss (1809, section 179)
69. Laplace (1774, Problem III)
70. Pearson (1905, p. 189)
71. Stigler (1986, p. 144)
72. Stigler (1978, p. 243)
73. Stigler (1978, p. 244)
74. Maxwell (1860, p. 23)

75. Jaynes, Edwin J.; *Probability Theory: The Logic of Science*, Ch 7 (<http://www.biba.inrialpes.fr/Jaynes/cc07s.pdf>)
76. Peirce, Charles S. (c. 1909 MS), *Collected Papers* v. 6, paragraph 327
77. Kruskal & Stigler (1997)
78. "Earliest uses... (entry STANDARD NORMAL CURVE)" (<http://jeff560.tripod.com/s.html>).

## Sources

- Aldrich, John; Miller, Jeff. "Earliest Uses of Symbols in Probability and Statistics" (<http://jeff560.tripod.com/stat.html>).
- Aldrich, John; Miller, Jeff. "Earliest Known Uses of Some of the Words of Mathematics" (<http://jeff560.tripod.com/mathword.html>). In particular, the entries for "bell-shaped and bell curve" (<http://jeff560.tripod.com/b.html>), "normal (distribution)" (<http://jeff560.tripod.com/n.html>), "Gaussian" (<http://jeff560.tripod.com/g.html>), and "Error, law of error, theory of errors, etc." (<http://jeff560.tripod.com/e.html>).
- Amari, Shun-ichi; Nagaoka, Hiroshi (2000). *Methods of Information Geometry*. Oxford University Press. ISBN 978-0-8218-0531-2.
- Bernardo, José M.; Smith, Adrian F. M. (2000). *Bayesian Theory*. Wiley. ISBN 978-0-471-49464-5.
- Bryc, Włodzimierz (1995). *The Normal Distribution: Characterizations with Applications*. Springer-Verlag. ISBN 978-0-387-97990-8.
- Casella, George; Berger, Roger L. (2001). *Statistical Inference* (2nd ed.). Duxbury. ISBN 978-0-534-24312-8.
- Cody, William J. (1969). "Rational Chebyshev Approximations for the Error Function". *Mathematics of Computation*. **23** (107): 631–638. doi:10.1090/S0025-5718-1969-0247736-4 (<https://doi.org/10.1090%2FS0025-5718-1969-0247736-4>).
- Cover, Thomas M.; Thomas, Joy A. (2006). *Elements of Information Theory*. John Wiley and Sons.
- de Moivre, Abraham (1738). *The Doctrine of Chances*. ISBN 978-0-8218-2103-9.
- Fan, Jianqing (1991). "On the optimal rates of convergence for nonparametric deconvolution problems" (<https://doi.org/10.1214/aos/1176348248>). *The Annals of Statistics*. **19** (3): 1257–1272. doi:10.1214/aos/1176348248 (<https://doi.org/10.1214%2Faos%2F1176348248>). JSTOR 2241949 (<https://www.jstor.org/stable/2241949>).
- Galton, Francis (1889). *Natural Inheritance* (<http://galton.org/books/natural-inheritance/pdf/galton-nat-inh-1up-clean.pdf>) (PDF). London, UK: Richard Clay and Sons.
- Galambos, Janos; Simonelli, Italo (2004). *Products of Random Variables: Applications to Problems of Physics and to Arithmetical Functions* (<https://archive.org/details/productsofrandom00gala>). Marcel Dekker, Inc. ISBN 978-0-8247-5402-0.
- Gauss, Carolo Friderico (1809). *Theoria motus corporum coelestium in sectionibus conicis Solem ambientium* (<https://archive.org/details/theoriamotuscor00gausgoog>) [*Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections*] (in Latin). English translation (<https://books.google.com/books?id=1TIAAAAAQAAJ>).
- Gould, Stephen Jay (1981). *The Mismeasure of Man* (first ed.). W. W. Norton. ISBN 978-0-393-01489-1.
- Halperin, Max; Hartley, Herman O.; Hoel, Paul G. (1965). "Recommended Standards for Statistical Symbols and Notation. COPSS Committee on Symbols and Notation". *The American Statistician*. **19** (3): 12–14. doi:10.2307/2681417 (<https://doi.org/10.2307%2F2681417>). JSTOR 2681417 (<https://www.jstor.org/stable/2681417>).
- Hart, John F.; et al. (1968). *Computer Approximations*. New York, NY: John Wiley & Sons, Inc. ISBN 978-0-88275-642-4.
- "Normal Distribution" ([https://www.encyclopediaofmath.org/index.php?title=Normal\\_Distribution](https://www.encyclopediaofmath.org/index.php?title=Normal_Distribution)), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Herrnstein, Richard J.; Murray, Charles (1994). *The Bell Curve: Intelligence and Class Structure in American Life*. Free Press. ISBN 978-0-02-914673-6.
- Huxley, Julian S. (1932). *Problems of Relative Growth*. London. ISBN 978-0-486-61114-3. OCLC 476909537 (<http://www.worldcat.org/oclc/476909537>).
- Johnson, Norman L.; Kotz, Samuel; Balakrishnan, Narayanaswamy (1994). *Continuous Univariate Distributions, Volume 1*. Wiley. ISBN 978-0-471-58495-7.
- Johnson, Norman L.; Kotz, Samuel; Balakrishnan, Narayanaswamy (1995). *Continuous Univariate Distributions, Volume 2*. Wiley. ISBN 978-0-471-58494-0.
- Karney, C. F. F. (2016). "Sampling exactly from the normal distribution". *ACM Transactions on Mathematical Software*. **42** (1): 3:1–14. arXiv:1303.6257 (<https://arxiv.org/abs/1303.6257>). doi:10.1145/2710016 (<https://doi.org/10.1145/2710016>)

10.1145%2F2710016).

- Kinderman, Albert J.; Monahan, John F. (1977). "Computer Generation of Random Variables Using the Ratio of Uniform Deviates". *ACM Transactions on Mathematical Software*. **3** (3): 257–260. doi:10.1145/355744.355750 (https://doi.org/10.1145%2F355744.355750).
- Krishnamoorthy, Kalimuthu (2006). *Handbook of Statistical Distributions with Applications*. Chapman & Hall/CRC. ISBN 978-1-58488-635-8.
- Kruskal, William H.; Stigler, Stephen M. (1997). Spencer, Bruce D. (ed.). *Normative Terminology: 'Normal' in Statistics and Elsewhere*. Statistics and Public Policy. Oxford University Press. ISBN 978-0-19-852341-3.
- Laplace, Pierre-Simon de (1774). "Mémoire sur la probabilité des causes par les événements" (http://gallica.bnf.fr/ark:/12148/bpt6k77596b/f32). *Mémoires de l'Académie Royale des Sciences de Paris (Savants étrangers)*, Tome 6: 621–656. Translated by Stephen M. Stigler in *Statistical Science* 1 (3), 1986: JSTOR 2245476 (https://www.jstor.org/stable/2245476).
- Laplace, Pierre-Simon (1812). *Théorie analytique des probabilités* (https://archive.org/details/thorieanalytiqu00laplgoog) [*Analytical theory of probabilities*].
- Le Cam, Lucien; Lo Yang, Grace (2000). *Asymptotics in Statistics: Some Basic Concepts* (second ed.). Springer. ISBN 978-0-387-95036-5.
- Leva, Joseph L. (1992). "A fast normal random number generator" (https://web.archive.org/web/20100716035328/http://saluc.engr.uconn.edu/refs/crypto/rng/leva92afast.pdf) (PDF). *ACM Transactions on Mathematical Software*. **18** (4): 449–453. CiteSeerX 10.1.1.544.5806 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.544.5806). doi:10.1145/138351.138364 (https://doi.org/10.1145%2F138351.138364). Archived from the original (http://saluc.engr.uconn.edu/refs/crypto/rng/leva92afast.pdf) (PDF) on July 16, 2010.
- Lexis, Wilhelm (1878). "Sur la durée normale de la vie humaine et sur la théorie de la stabilité des rapports statistiques". *Annales de Démographie Internationale*. Paris. II: 447–462.
- Lukacs, Eugene; King, Edgar P. (1954). "A Property of Normal Distribution" (https://doi.org/10.1214/aoms/1177728796). *The Annals of Mathematical Statistics*. **25** (2): 389–394. doi:10.1214/aoms/1177728796 (https://doi.org/10.1214%2Faoms%2F1177728796). JSTOR 2236741 (https://www.jstor.org/stable/2236741).
- McPherson, Glen (1990). *Statistics in Scientific Investigation: Its Basis, Application and Interpretation* (https://archive.org/details/statisticsinscie0000mcph). Springer-Verlag. ISBN 978-0-387-97137-7.
- Marsaglia, George; Tsang, Wai Wan (2000). "The Ziggurat Method for Generating Random Variables" (https://doi.org/10.18637/jss.v005.i08). *Journal of Statistical Software*. **5** (8). doi:10.18637/jss.v005.i08 (https://doi.org/10.18637%2Fjss.v005.i08).
- Marsaglia, George (2004). "Evaluating the Normal Distribution" (https://doi.org/10.18637/jss.v011.i04). *Journal of Statistical Software*. **11** (4). doi:10.18637/jss.v011.i04 (https://doi.org/10.18637%2Fjss.v011.i04).
- Maxwell, James Clerk (1860). "V. Illustrations of the dynamical theory of gases. — Part I: On the motions and collisions of perfectly elastic spheres". *Philosophical Magazine*. Series 4. **19** (124): 19–32. doi:10.1080/14786446008642818 (https://doi.org/10.1080%2F14786446008642818).
- Monahan, J. F. (1985). "Accuracy in random number generation" (https://doi.org/10.1090/S0025-5718-1985-0804945-X). *Mathematics of Computation*. **45** (172): 559–568. doi:10.1090/S0025-5718-1985-0804945-X (https://doi.org/10.1090%2FS0025-5718-1985-0804945-X).
- Patel, Jagdish K.; Read, Campbell B. (1996). *Handbook of the Normal Distribution* (2nd ed.). CRC Press. ISBN 978-0-8247-9342-5.
- Pearson, Karl (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space" (http://stat.smmu.edu.cn/history/pearson1901.pdf) (PDF). *Philosophical Magazine*. **6**. **2** (11): 559–572. doi:10.1080/14786440109462720 (https://doi.org/10.1080%2F14786440109462720).
- Pearson, Karl (1905). "'Das Fehlergesetz und seine Verallgemeinerungen durch Fechner und Pearson'. A rejoinder". *Biometrika*. **4** (1): 169–212. doi:10.2307/2331536 (https://doi.org/10.2307%2F2331536). JSTOR 2331536 (https://www.jstor.org/stable/2331536).
- Pearson, Karl (1920). "Notes on the History of Correlation" (https://zenodo.org/record/1431597/files/article.pdf) (PDF). *Biometrika*. **13** (1): 25–45. doi:10.1093/biomet/13.1.25 (https://doi.org/10.1093%2Fbiomet%2F13.1.25). JSTOR 2331722 (https://www.jstor.org/stable/2331722).
- Rohrbasser, Jean-Marc; Véron, Jacques (2003). "Wilhelm Lexis: The Normal Length of Life as an Expression of the 'Nature of Things'" (http://www.persee.fr/web/revues/home/prescript/article/pop\_1634-2941\_2003\_num\_58\_3\_18444). *Population*. **58** (3): 303–322. doi:10.3917/pope.303.0303 (https://doi.org/10.3917%2Fpope.303.0303).
- Shore, H (1982). "Simple Approximations for the Inverse Cumulative Function, the Density Function and the Loss Integral of the Normal Distribution". *Journal of the Royal Statistical Society. Series C (Applied Statistics)*. **31** (2):

- 108–114. doi:10.2307/2347972 (<https://doi.org/10.2307%2F2347972>). JSTOR 2347972 (<https://www.jstor.org/stable/2347972>).
- Shore, H (2005). "Accurate RMM-Based Approximations for the CDF of the Normal Distribution". *Communications in Statistics – Theory and Methods*. **34** (3): 507–513. doi:10.1081/sta-200052102 (<https://doi.org/10.1081%2Fsta-200052102>).
  - Shore, H (2011). "Response Modeling Methodology". *WIREs Comput Stat*. **3** (4): 357–372. doi:10.1002/wics.151 (<https://doi.org/10.1002%2Fwics.151>).
  - Shore, H (2012). "Estimating Response Modeling Methodology Models". *WIREs Comput Stat*. **4** (3): 323–333. doi:10.1002/wics.1199 (<https://doi.org/10.1002%2Fwics.1199>).
  - Stigler, Stephen M. (1978). "Mathematical Statistics in the Early States" (<https://doi.org/10.1214/aos/1176344123>). *The Annals of Statistics*. **6** (2): 239–265. doi:10.1214/aos/1176344123 (<https://doi.org/10.1214%2Faos%2F1176344123>). JSTOR 2958876 (<https://www.jstor.org/stable/2958876>).
  - Stigler, Stephen M. (1982). "A Modest Proposal: A New Standard for the Normal". *The American Statistician*. **36** (2): 137–138. doi:10.2307/2684031 (<https://doi.org/10.2307%2F2684031>). JSTOR 2684031 (<https://www.jstor.org/stable/2684031>).
  - Stigler, Stephen M. (1986). *The History of Statistics: The Measurement of Uncertainty before 1900* (<https://archive.org/details/historyofstatist00stig>). Harvard University Press. ISBN 978-0-674-40340-6.
  - Stigler, Stephen M. (1999). *Statistics on the Table*. Harvard University Press. ISBN 978-0-674-83601-3.
  - Walker, Helen M. (1985). "De Moivre on the Law of Normal Probability" (<http://www.york.ac.uk/depts/maths/histstat/demoivre.pdf>) (PDF). In Smith, David Eugene (ed.). *A Source Book in Mathematics*. Dover. ISBN 978-0-486-64690-9.
  - Wallace, C. S. (1996). "Fast pseudo-random generators for normal and exponential variates". *ACM Transactions on Mathematical Software*. **22** (1): 119–127. doi:10.1145/225545.225554 (<https://doi.org/10.1145%2F225545.225554>).
  - Weisstein, Eric W. "Normal Distribution" (<http://mathworld.wolfram.com/NormalDistribution.html>). MathWorld.
  - West, Graeme (2009). "Better Approximations to Cumulative Normal Functions" ([http://www.wilmott.com/pdfs/090721\\_west.pdf](http://www.wilmott.com/pdfs/090721_west.pdf)) (PDF). *Wilmott Magazine*: 70–76.
  - Zelen, Marvin; Severo, Norman C. (1964). *Probability Functions (chapter 26)* ([http://www.math.sfu.ca/~cbm/aands/page\\_931.htm](http://www.math.sfu.ca/~cbm/aands/page_931.htm)). *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, by Abramowitz, M.; and Stegun, I. A.: National Bureau of Standards. New York, NY: Dover. ISBN 978-0-486-61272-0.

## External links

- "Normal distribution" ([https://www.encyclopediaofmath.org/index.php?title=Normal\\_distribution](https://www.encyclopediaofmath.org/index.php?title=Normal_distribution)), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Normal distribution calculator (<https://www.hackmath.net/en/calculator/normal-distribution>), More powerful calculator (<https://keisan.casio.com/exec/system/1180573188>)

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Normal\\_distribution&oldid=978806832](https://en.wikipedia.org/w/index.php?title=Normal_distribution&oldid=978806832)"

This page was last edited on 17 September 2020, at 01:50 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

**Source:** [https://en.wikipedia.org/wiki/Box\\_plot](https://en.wikipedia.org/wiki/Box_plot)

A boxplot is a standardized way of displaying the dataset based on a five-number summary: the minimum, the maximum, the sample median, and the first and third quartiles.

**Minimum** : the lowest data point excluding any outliers.

**Maximum** : the largest data point excluding any outliers.

**Median ( $Q_2$  / 50th percentile)** : the middle value of the dataset.

**First quartile ( $Q_1$  / 25th percentile)** : also known as the *lower quartile*  $q_n(0.25)$ , is the median of the lower half of the dataset.

**Third quartile ( $Q_3$  / 75th percentile)** : also known as the *upper quartile*  $q_n(0.75)$ , is the median of the upper half of the dataset.<sup>[4]</sup>

An important element used to construct the box plot by determining the minimum and maximum data values feasible, but is not part of the aforementioned five-number summary, is the interquartile range or IQR denoted below:

**Interquartile range (IQR)** : is the distance between the upper and lower quartiles.

A boxplot is constructed of two parts, a box and a set of whiskers shown in Figure 2. The lowest point is the minimum of the data set and the highest point is the maximum of the data set. The box is drawn from  $Q_1$  to  $Q_3$  with a horizontal line drawn in the middle to denote the median.

The same data set can also be represented as a boxplot shown in Figure 3. From above the upper quartile, a distance of 1.5 times the IQR is measured out and a whisker is drawn up to the largest observed point from the dataset that falls within this distance. Similarly, a distance of 1.5 times the IQR is measured out below the lower quartile and a whisker is drawn up to the lower observed point from the dataset that falls within this distance. All other observed points are plotted as outliers.

However, the whiskers can represent several possible alternative values, among them:

- the minimum and maximum of all of the data (as in figure 2)
- one standard deviation above and below the mean of the data
- the 9th percentile and the 91st percentile
- the 2nd percentile and the 98th percentile.

Any data not included between the whiskers should be plotted as an outlier with a dot, small circle, or star, but occasionally this is not done.

Some box plots include an additional character to represent the mean of the data.

On some box plots a crosshatch is placed on each whisker, before the end of the whisker.

Rarely, box plots can be presented with no whiskers at all.

Because of this variability, it is appropriate to describe the convention being used for the whiskers and outliers in the caption for the plot.

The unusual percentiles 2%, 9%, 91%, 98% are sometimes used for whisker cross-hatches and whisker ends to show the seven-number summary. If the data are normally distributed, the locations of the seven marks on the box plot will be equally spaced.

## Boxplot with whiskers from minimum to maximum

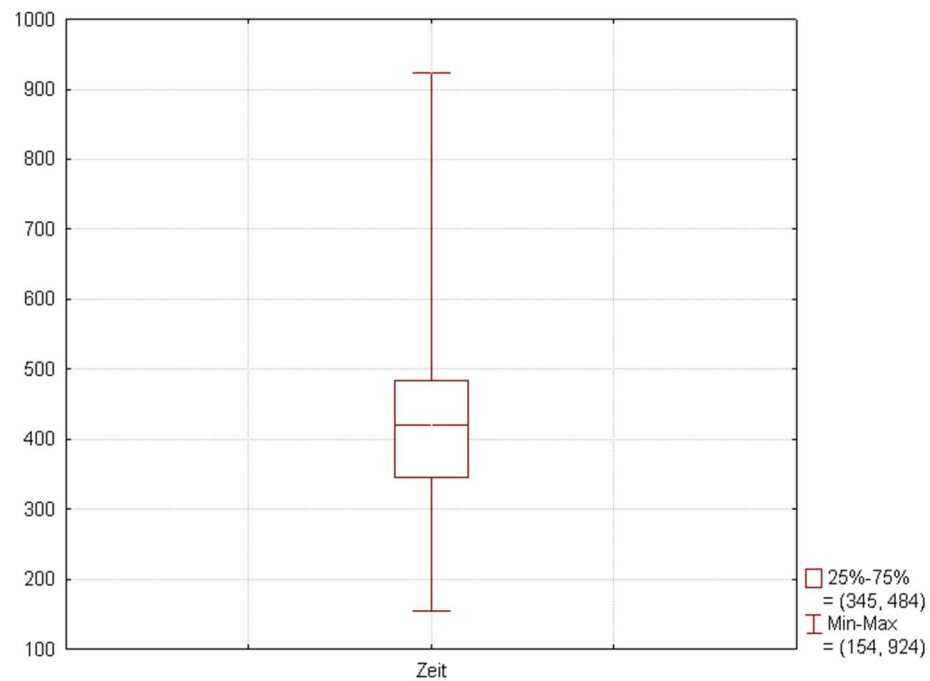
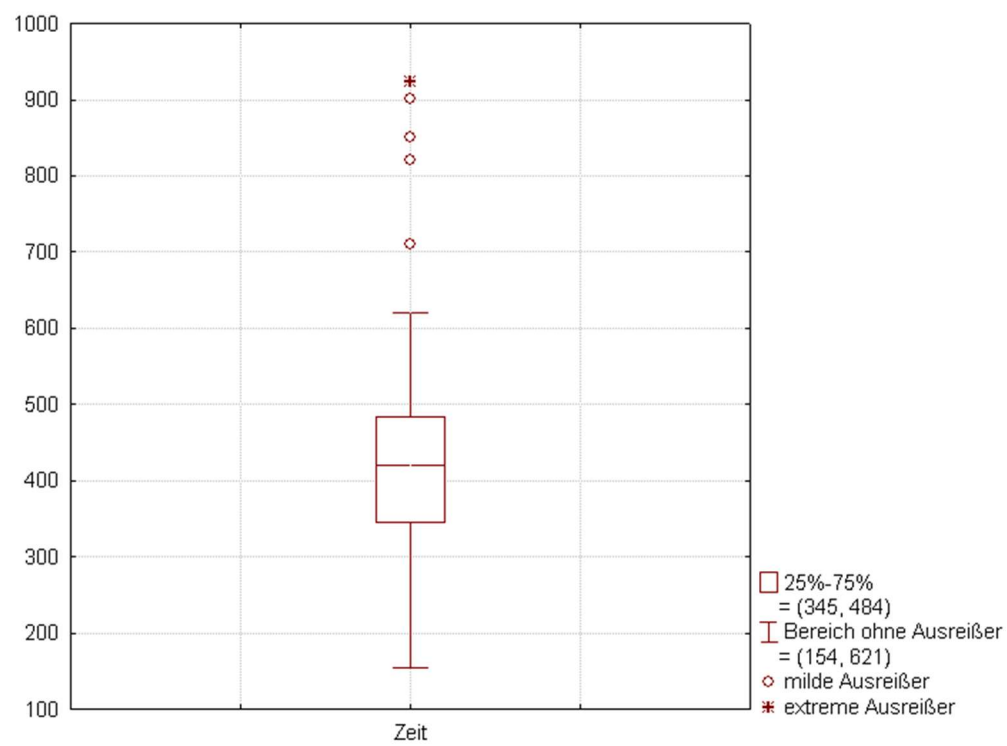


Figure 3. Same Boxplot with whiskers with maximum 1.5 IQR



## Example without outliers

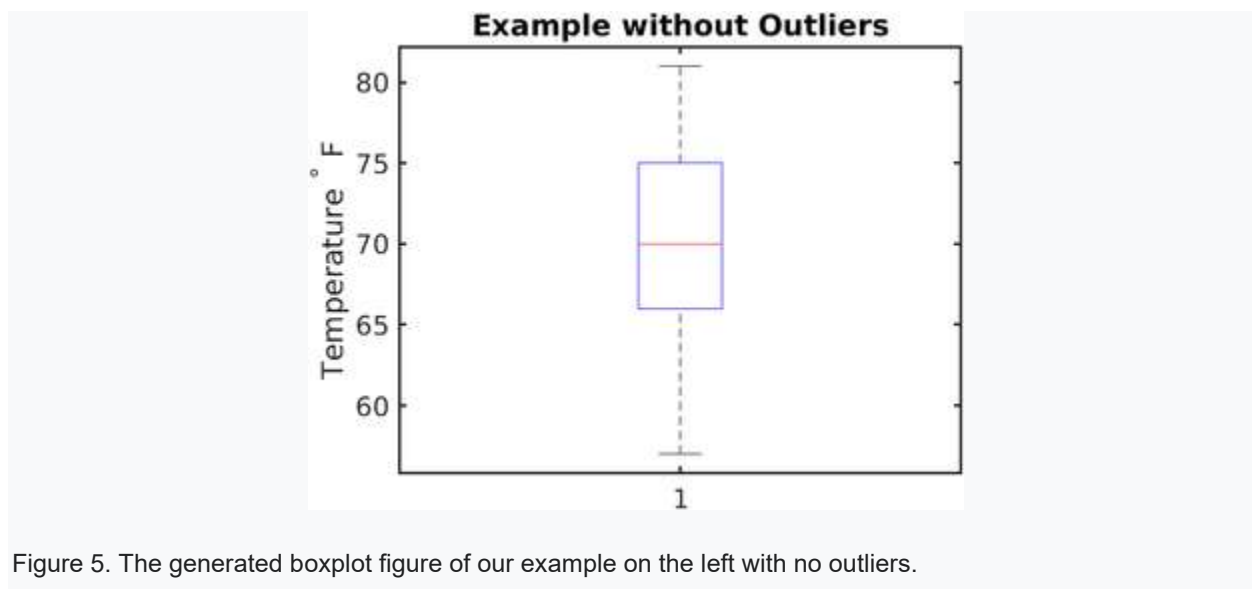


Figure 5. The generated boxplot figure of our example on the left with no outliers.

A series of hourly temperatures were measured throughout the day in degrees Fahrenheit. The recorded values are listed in order as follows: 50, 50, 55, 58, 63, 66, 66, 67, 67, 68, 69, 70, 70, 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 81.

A box plot of the data can be generated by calculating five relevant values: minimum, maximum, median, first quartile, and third quartile.

The minimum is the smallest number of the set. In this case, the minimum day temperature is 50 °F.

The maximum is the largest number of the set. In this case, the maximum day temperature is 81 °F.

The median is the "middle" number of the ordered set. This means that there are exactly 50% of the elements less than the median and 50% of the elements greater than the median. The median of this ordered set is 70 °F.

The first quartile value is the number that marks one quarter of the ordered set. In other words, there are exactly 25% of the elements that are less than the first quartile and exactly 75% of the elements that are greater. The first quartile value can easily be determined by finding the "middle" number between the minimum and the median. For the hourly temperatures, the "middle" number between 50 °F and 70 °F is 66 °F.

The third quartile value is the number that marks three quarters of the ordered set. In other words, there are exactly 75% of the elements that are less than the first quartile and 25% of the elements that are greater. The third quartile value can be easily determined by finding the "middle" number between the median and the maximum. For the hourly temperatures, the "middle" number between 70 °F and 81 °F is 75 °F.

The upper whisker of the box plot is the largest dataset number smaller than 1.5IQR above the third quartile. Here, 1.5IQR above the third quartile is 88.5 °F and the maximum is 81 °F. Therefore, the upper whisker is drawn at the value of the maximum, 81 °F.

Similarly, the lower whisker of the box plot is the smallest dataset number larger than 1.5IQR below the first quartile. Here, 1.5IQR below the first quartile is 52.5 °F and the minimum is 50 °F. Therefore, the lower whisker is drawn at the value of the smallest dataset number larger than 52.5 °F, 55 °F.

## Example with outliers

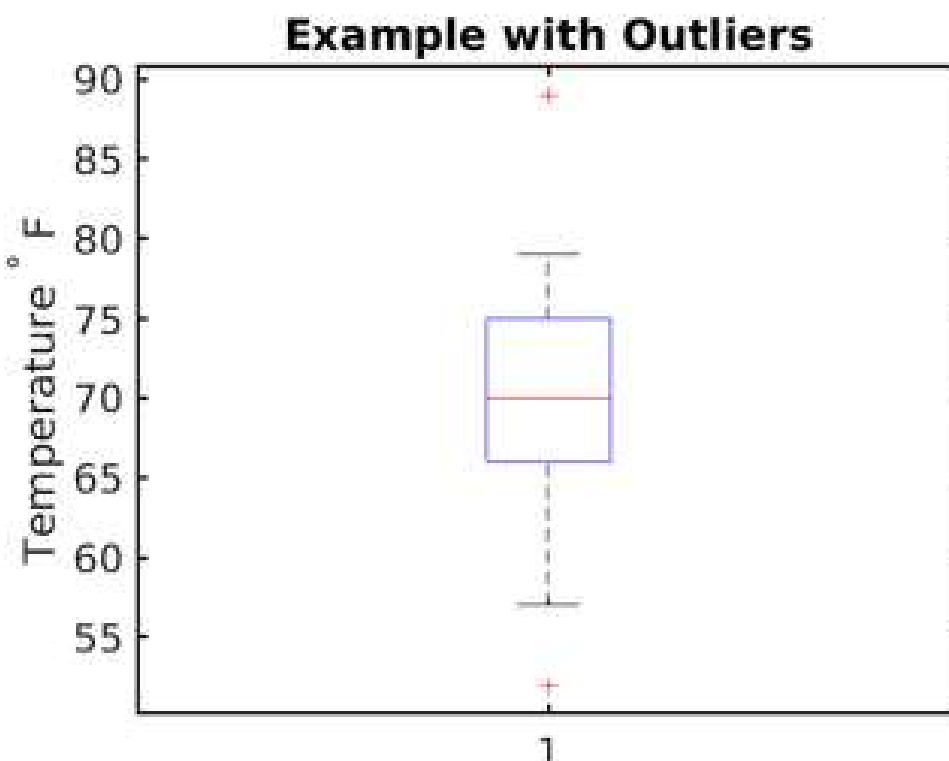


Figure 6. The generated boxplot of our example on the left with outliers.

Above is an example without outliers. Here is a followup example with outliers:

The ordered set is: 52, 57, 57, 58, 63, 66, 66, 67, 67, 68, 69, 70, 70, 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 89.

In this example, only the first and the last number are changed. The median, third quartile, and first quartile remain the same.

In this case, the maximum is 89 °F and 1.5IQR above the third quartile is 88.5 °F. The maximum is greater than 1.5IQR plus the third quartile, so the maximum is an outlier. Therefore, the upper whisker is drawn at the greatest value smaller than 1.5IQR above the third quartile, which is 79 °F.

Similarly, the minimum is 52 °F and 1.5IQR below the first quartile is 52.5 °F. The minimum is smaller than 1.5IQR minus the first quartile, so the minimum is also an outlier. Therefore, the lower whisker is drawn at the smallest value greater than 1.5IQR below the first quartile, which is 57 °F.

#### In the case of large datasets [\[ edit \]](#)

##### General equation to compute empirical quantiles [\[ edit \]](#)

$$q_n(p) = x_{(k)} + \alpha(x_{(k+1)} - x_{(k)})$$

$$\text{with } k = [p(n+1)] \text{ and } \alpha = p(n+1) - k$$

Using the example from above with 24 data points, meaning  $n = 24$ , one can also calculate the median, first and third quartile mathematically vs. visually.

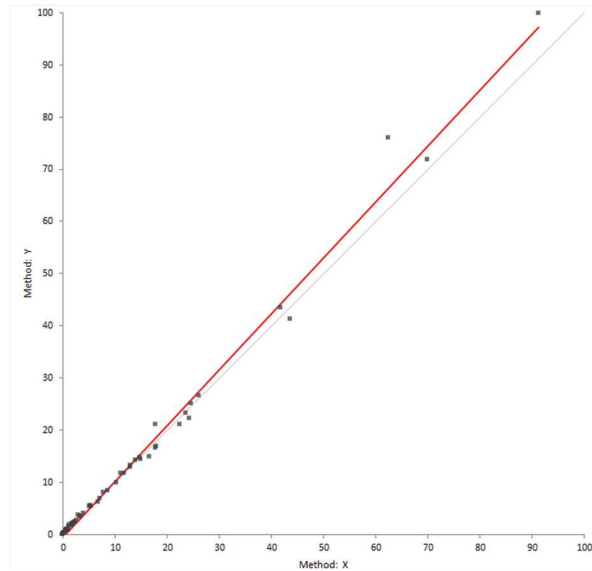
$$\text{Median : } q_n(0.5) = q_{(12)} + (0.5 \cdot 25 - 12) \cdot (x_{(13)} - x_{(12)}) = 70 + (0.5 \cdot 25 - 12) \cdot (70 - 70) = 70$$

$$\text{First quartile : } q_n(0.25) = q_{(6)} + (0.25 \cdot 25 - 6) \cdot (x_{(7)} - x_{(6)}) = 66 + (0.25 \cdot 25 - 6) \cdot (66 - 66) = 66$$

$$\text{Third quartile : } q_n(0.75) = q_{(18)} + (0.75 \cdot 25 - 18) \cdot (x_{(19)} - x_{(18)}) = 75 + (0.75 \cdot 25 - 18) \cdot (75 - 75) = 75$$

# Scatter plot (method comparison)

A scatter plot shows the relationship between two methods.



The scatter plot shows measured values of the reference or comparison method on the horizontal axis, against the test method on the vertical axis.

The relationship between the methods may indicate a constant, or proportional bias, and the variability in the measurements across the measuring interval. If the points form a constant-width band, the method has a constant standard deviation (constant SD). If the points form a band that is narrower at small values and wider at large values, there is a constant relationship between the standard deviation and value, and the method has constant a coefficient of variation (CV). Some measurement procedures exhibit constant SD in the low range and constant CV in the high range.

If both methods measure on the same scale, a gray identity line shows ideal agreement and is useful for comparing the relationship against.

## 2.5: Correlation and Causation, Scatter Plots

The strength of a relationship between two variables is called **correlation**. Variables that are strongly related to each other have strong correlation. However, if two variables are correlated it does not mean that one variable caused the other variable to occur. The above example from the

Planters Cocktail Peanuts label is an example of this. There is a strong correlation between eating a diet that is low in saturated fat and cholesterol and heart disease. But that correlation does not mean that eating a diet that is low in saturated fat and cholesterol will cause your risk of heart disease to go down. There could be many different variables that could cause both variables in question to go down or up. One example is that a person's genetic makeup could make them not want to eat fatty food and also not develop heart disease. No matter how strong a correlation is between two variables, you can never know for sure if one variable causes the other variable to occur without conducting experimentation. The only way to find out if eating a diet low in saturated fat and cholesterol actually lowers the risk of heart disease is to do an experiment. This is where you tell one group of people that they have to eat a diet low in saturated fat and cholesterol and another group of people that they have to eat a diet high in saturated fat and cholesterol, and then observe what happens to both groups over the years. You cannot morally do this experiment, so there is no way to prove the statement. That is why the word "may" is in the statement. We see many correlations like this one. Always be sure not to make a correlation statement into a causation statement.

### Example 2.5.12.5.1: Correlation vs Causation

For each of the following scenarios answer the question and give an example of another variable that could explain the correlation.

1. There is a negative correlation between number of children a woman has and her life expectancy. Does that mean that having children causes a woman to die earlier?

A correlation between two variables does not mean that one causes the other. A possible cause for both variables could be better health care. If there is better health care, then life expectancy goes up, and also with better health care birth control is more readily available.

2. There is a positive correlation between ice cream sales and the number of drownings at the beach. Does that mean that eating ice cream can cause a person to drown?

A correlation between two variables does not mean that one causes the other. The cause for both could be that the temperature is going up. The higher the temperature, the more likely someone will buy ice cream and the more people at the beach.

3. There is a correlation between waist measures and wrist measures. Does this mean that your waist measurement causes your wrist measurement to change?

A correlation between two variables does not mean that one causes the other. The cause of both could be a person's genetics, eating habits, exercise habits, etc.

How do we tell if there is a correlation between two variables? The easiest way is to graph the two variables together as ordered pairs on a graph called a **scatter plot**. To create a scatter plot, consider that one variable is the independent variable and the other is the dependent variable. This means that the dependent variable depends on the independent variable. We usually set up these two variables as ordered pairs where the independent variable is first and the dependent variable

is second. Thus, when graphed, the independent variable is graphed along the horizontal axis and the dependent variable is graphed along the vertical axis. You do not connect the dots after plotting these ordered pairs. Instead look to see if there is a pattern, such as a line, that fits the data well. Here are some examples of scatter plots and how strong the linear correlation is between the two variables.

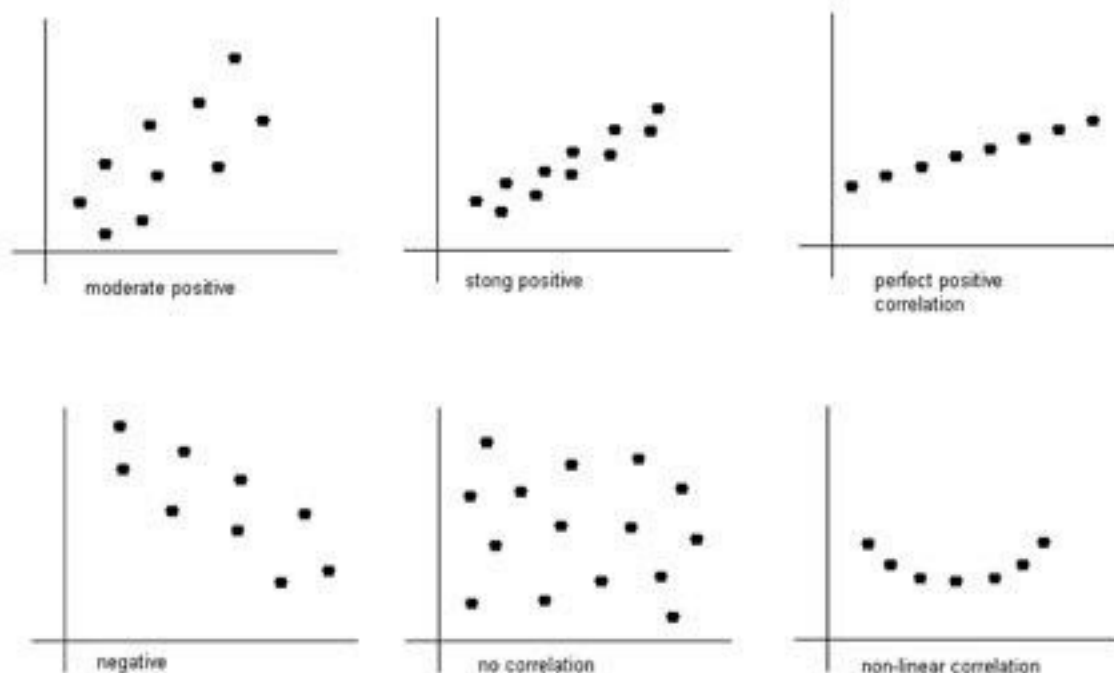


Figure 2.5.12.5.1: Scatter Plots Showing Types of Linear Correlation

# Null Replacement

**Source:** [https://docs.teradata.com/reader/D8pLcnswlJJUTD9q0xD0Buw/Rgr~tkmosOxOreWqyseB\\_A](https://docs.teradata.com/reader/D8pLcnswlJJUTD9q0xD0Buw/Rgr~tkmosOxOreWqyseB_A)

## Purpose

NULL value replacement is offered as a transformation function. A literal value, the mean, median, mode, or an imputed value joined from another table can be used as the replacement value. The median value can be requested with or without averaging of two middle values when there is an even number of values.

Literal value replacement is supported for numeric, character, and date data types. Mean value replacement is supported for columns of numeric type or date type. Median without averaging, mode, and imputed value replacement are valid for any supported type. Median with averaging is supported only for numeric and date type.

Null can be checked by below query:-

```
SELECT Salary
FROM Employee
WHERE DeptNo IS NULL;
```

## Examples

These examples demonstrate the Null Replacement transformation. To run the provided examples, the `td_analyze` function must be installed in a database called `twm` and the TWM tutorial data must be installed in the `twm_source` database.

The first example operates on numeric data.

```
call
twm.td_analyze('vartran','database=twm_source;tablename=twm_customer;keycolumns=cust
_id>nullreplacement={nullstyle(literal,0),columns(age,inc)}{nullstyle(mean),columns(ag
e/age1)}{nullstyle(median),columns(age/age2)}{nullstyle(medianwithoutaveraging),columns(ag
e/age3)}{nullstyle(mode),columns(age/age4)}{nullstyle(imputed,twm_customer_analysis),colu
mns(income)};');
```

This example operates on date and character type data.

```
call
twm.td_analyze('vartran','database=twm_source;tablename=twm_credit_acct;keycolumns=cus
t_id>nullreplacement={nullstyle(literal,DATE 1995-12-
23),columns(acct_end_date/date1)}{nullstyle(literal,U),columns(account_active/char1)}{nullstyl
e(mean),columns(acct_end_date/date2)}{nullstyle(median),columns(acct_end_date/date2A)}{n
ullstyle(medianwithoutaveraging),columns(acct_end_date/date3)}{nullstyle(mode),columns(ac
ct_end_date/date4)}{nullstyle(imputed,twm_checking_acct),columns(acct_end_date/date5)}{n
ullstyle(medianwithoutaveraging),columns(account_active/char2)}{nullstyle(mode),columns(ac
count_active/char3)}{nullstyle(imputed,twm_checking_acct),columns(account_active/char4)};')
```

## Recode

[https://docs.teradata.com/reader/D8pLcnsWIJUTD9q0xD0Buw/HTwi8XblqKAnLR\\_hQpNG9A](https://docs.teradata.com/reader/D8pLcnsWIJUTD9q0xD0Buw/HTwi8XblqKAnLR_hQpNG9A)

Recoding a categorical data column is done to *re-express* existing values of a column (variable) into a new *coding scheme* or to correct data quality problems and focus an analysis on a particular value. It allows for mapping individual values, NULL values, or any number of remaining values (ELSE option) to a new value, a NULL value or the same value.

Recoding supports character, numeric, and date type columns. If date values are entered, the keyword DATE must precede the date value, and do not enclose in single quotes.

The following example demonstrates the Recode transformation.

```
call
twm.td_analyze('vartran','database=twm_source;tablename=twm_customer;recode={recodeval
ues(M/SAME,F/f),recodeother(NULL),columns(gender)}{recodevalues(1/SAME,2/NULL,3/6,4/4,
NULL/NULL),recodeother(NULL),columns(marital_status)}{recodevalues(F/f,null/0),recodeother
(same),columns(gender/gender2)}{recodevalues(0/0,1/1,2/1,3/1,4/1,5/1),recodeother(0),colu
mns(nbr_children,years_with_bank)};')
```

## Rescale

[https://docs.teradata.com/reader/D8pLcnsWIJUTD9q0xD0Buw/aIIzUpU7O1Jvsq~9\\_NUfWw](https://docs.teradata.com/reader/D8pLcnsWIJUTD9q0xD0Buw/aIIzUpU7O1Jvsq~9_NUfWw)

### Purpose

Rescaling limits the upper and lower boundaries of the data in a continuous numeric column using a linear rescaling function based on maximum and minimum data values. Rescale is useful with algorithms that require or work better with data within a certain range. Rescale is only valid on numeric columns, and not columns of type date.

You can supply new minimum and maximum values to form new variable boundaries. If only the lower boundary is supplied, the variable is aligned to this value; if only an upper boundary value is specified, the variable is aligned to that value. If a requested column has a constant value (max and min are the same), then the transformation fails with an SQL error.

**The following example demonstrates the Rescale transformation.**

```
call
twm.td_analyze('vartran','database=twm_source;tablename=twm_customer;rescale={rescaleb
ounds(lowerbound/0,upperbound/1),columns(income/inc,age)}{rescalebounds(upperbound/1),
columns(income/income1,age/age1)}{rescalebounds(lowerbound/0),columns(income/income2
,age/age2)};');
```

## Sigmoid

<https://docs.teradata.com/reader/D8pLcnsWlJUTD9q0xD0BuW/h8ApE8PwhJ6HpZ0GaQUhZQ>

### Purpose

A Sigmoid transformation provides rescaling of continuous numeric data in a more sophisticated way than the Rescaling transformation function. In a Sigmoid transformation a numeric column is transformed using a type of sigmoid or s-shaped function. The logit function produces a continuously increasing value between 0 and 1. The modified logit function is twice the logit minus 1 and produces a value between -1 and 1. The hyperbolic tangent function also produces a value between -1 and 1. These non-linear transformations are more useful in data mining than a linear Rescaling transformation.

For absolute values of x greater than or equal to 36, the value of the sigmoid function is effectively 1 for positive arguments or 0 for negative arguments, within about 15 digits of significance.

The Sigmoid transformation is supported for numeric columns only, not date columns. The only required parameter for the Sigmoid transformation is `columns`. The `datatype` parameter controls the output data type. The `sigmoidstyle` parameter is also specifies the style of sigmoid function.

The following example demonstrates the Sigmoid transformation.

```
call
twm.td_analyze('vartran','database=twm_source;tablename=twm_customer;sigmoid={sigmoid
style(logit),columns(cust_id,age,income)}{sigmoidstyle(modifiedlogit),columns(cust_id/cid2,age
/age2,income/inc2)}{sigmoidstyle(tanh),columns(cust_id/cid3,age/age3,income/inc3)};');
```

## BAD QUALITY DATA CHECK

(source-: Teradata SQL Function Expression and Predicates document)

### EXIST and NOT EXIST PREDICATES

EXISTS predicate tests the existence of specified rows of a subquery. In general, EXISTS can be used to replace comparisons with IN and NOT EXISTS can be used to replace comparisons with NOT IN. However, the reverse is not true. Some problems can be solved only by using EXISTS and/or NOT EXISTS predicate.

```
SELECT SName, SNo
FROM student s
WHERE EXISTS
(SELECT *
FROM department d
WHERE EXISTS
(SELECT *
FROM course c, registration r, class cl
WHERE c.Dept=d.Dept
AND c.CNo=r.CNo
AND s.SNo=r.SNo
AND r.CNo=cl.CNo
AND r.Sec=cl.Sec));
```

```
SELECT SName, SNo
FROM student s
WHERE NOT EXISTS
(SELECT *
FROM department d
WHERE d.Dept IN
(SELECT Dept
FROM course) AND NOT EXISTS
(SELECT *
FROM course c, registration r, class cl
WHERE c.Dept=d.Dept
AND c.CNo=r.CNo
AND s.SNo=r.SNo
```

```
AND r.CNo=cl.CNo
AND r.Sec=cl.Sec));
```

## IN/NOT IN

### Purpose

Tests the existence of the value of an expression or expression list in a comparable set in one of two ways:

- Compares the value of an expression with values in an explicit list of literals.
- Compares values in a list of expressions with values and in a set of corresponding expressions in a subquery.

The following statement searches for the names of all employees who work in Atlanta.

```
SELECT Name
FROM Employee
WHERE DeptNo IN
(SELECT DeptNo
FROM Department
WHERE Loc = 'ATL');
```

### Example: Using IN/NOT IN with a List of Literals

This example shows the behavior of IN/NOT IN with a list of literals.

Consider the following table definition and contents:

```
CREATE TABLE t (x INTEGER);
INSERT t (1);
INSERT t (2);
INSERT t (3);
INSERT t (4);
INSERT t (5);
```

Query	Result
SELECT * FROM t WHERE x IN (1,2)	1, 2
SELECT * FROM t WHERE x IN ANY (1,2)	1, 2
SELECT * FROM t WHERE NOT (x NOT IN (1,2))	1, 2
SELECT * FROM t WHERE x NOT IN (1,2)	3, 4, 5
SELECT * FROM t WHERE x NOT IN ALL (1,2)	3, 4, 5
SELECT * FROM t WHERE NOT (x IN (1, 2))	3, 4, 5
SELECT * FROM t WHERE NOT (x IN ANY (1,2))	3, 4, 5
SELECT * FROM t WHERE x IN (3 TO 5)	3, 4, 5
SELECT * FROM t WHERE x NOT IN SOME (1, 2)	1, 2, 3, 4, 5
SELECT * FROM t WHERE x IN (1, 2 TO 4, 5)	1, 2, 3, 4, 5
SELECT * FROM t WHERE x IN ALL (1,2)	no rows
SELECT * FROM t WHERE NOT (x NOT IN SOME (1,2))	no rows
SELECT * FROM t WHERE x NOT IN (1 TO 5)	no rows

The data can be deleted which is not required using -:

```
DELETE FROM t WHERE NOT (x IN (1, 2))
```

## Basic Sampling (Weighted)

<https://docs.teradata.com/reader/JtLhZxnZVIJAs8pZG1VVfg/EE9WkcAqwaJmGDLD32QShw>

This example uses basic sampling to select a sample of 10 rows, weighted by car weight. Because the function call includes the Seed and SeedColumn arguments, it always produces the same output from the same input.

The sampling can be done to avoid data quality issues.

```
SELECT * FROM RandomSample (  
  ON (SELECT 1) PARTITION BY 1  
  InputTable ('fs_input')  
  SamplingMode ('basic')  
  NumSample ('10')  
  WeightColumn ('wt')  
  Seed ('1')  
  SeedColumn ('model')  
) ORDER BY 1, 2, 3;
```

## MultiCaseMatch (ML Engine)

The MultiCaseMatch function extends the capability of the SQL CASE statement by supporting matches to multiple criteria in a single row.

When SQL CASE finds a match, it outputs the result and immediately proceeds to the next row without searching for more matches in the current row.

The MultiCaseMatch function iterates through the input data set only once and outputs matches whenever a match occurs. If multiple matches occur for a given input row, the function outputs one output row for each match.

Use the MultiCaseMatch function when the conditions in your CASE statement do not form a mutually exclusive set.

### MultiCaseMatch Syntax

#### Version 1.5

```
SELECT * FROM MultiCaseMatch (
  ON (SELECT t.*, condition AS case [,...] FROM { table | view | (query) } AS t)
  USING
  Labels ('case AS "label"' [,...])
) AS alias;
```

### MultiCaseMatch Syntax Elements

#### Labels

Specify a label for each *case*. Each case corresponds to a *condition*, which is a SQL predicate that includes input column names. When an input value satisfies *condition*, that is a match, and the function outputs the input row and the corresponding label.

### MultiCaseMatch Input

#### Input Table Schema

Column	Data Type	Description
<i>column_in_condition</i>	Any	[Column appears once for each column specified in a <i>condition</i> .]
<i>other_column</i>	Any	[Column appears zero or more times.] Column to copy to output table.

## MultiCaseMatch Output

### Output Table Schema

Column	Data Type	Description
<i>column_in_condition</i>	Same as in input table	[Column appears once for each column specified in a <i>condition</i> .] ] Column copied from input table.
labels	VARCHAR	[Column appears once for each matching label.] Labels that correspond to case that <i>column_in_condition</i> value matches.

## MultiCaseMatch Example

This example labels people with the age groups to which they belong, which overlap:

Age Group	Description
infant	Younger than 1 year
toddler	1-2 years, inclusive
kid	2-12 years, inclusive
teenager	13-19 years, inclusive
young adult	16-25 years, inclusive
adult	21-40 years, inclusive
middle-aged person	35-60 years, inclusive
senior citizen	60 years or older

### Input

people\_age

id	name	age
1	John	0.5
2	Freddy	2
3	Marie	6
4	Tom Sawyer	17
5	Becky Thatcher	16
6	Philip	22
7	Joseph	25

id	name	age
8	Roger	35
9	Natalie	30
10	Henry	40
11	George	50
12	Sir William	65

## SQL Call

```

SELECT * FROM MultiCaseMatch (
ON (SELECT t.*,
    (case when t.age < 1 THEN '1' ELSE '0' END) AS case1,
    (case when t.age >= 1 AND t.age <= 2 THEN '1' ELSE '0' END) AS case2,
    (case when t.age >= 2 AND t.age <= 12 THEN '1' ELSE '0' END) AS case3,
    (case when t.age >= 13 AND t.age <= 19 THEN '1' ELSE '0' END) AS case4,
    (case when t.age >= 16 AND t.age <= 25 THEN '1' ELSE '0' END) AS case5,
    (case when t.age >= 21 AND t.age <= 40 THEN '1' ELSE '0' END) AS case6,
    (case when t.age >= 35 AND t.age <= 60 THEN '1' ELSE '0' END) AS case7,
    (case when t.age >= 60 THEN '1' ELSE '0' END) AS case8
FROM people_age AS t)
USING
    LABELS (
        'case1 AS "infant"',
        'case2 AS "toddler"',
        'case3 AS "kid"',
        'case4 AS "teenager"',
        'case5 AS "young adult"',
        'case6 AS "adult"',
        'case7 AS "middle aged person"',
        'case8 AS "senior citizen"')
) AS dt;

```

## Output

Several people have two labels. For example, Freddy is both a toddler and a kid, and Tom Sawyer and Becky Thatcher are both teenagers and young adults.

id	name	age	labels
7	joseph	25.0	young adult
6	philip	22.0	young adult
7	joseph	25.0	adult

6 philip	22.0 adult
12 sir william	65.0 senior citizen
11 george	50.0 middle aged person
4 tom sawyer	17.0 teenager
9 natalie	30.0 adult
4 tom sawyer	17.0 young adult
10 henry	40.0 adult
5 becky thatcher	16.0 teenager
10 henry	40.0 middle aged person
5 becky thatcher	16.0 young adult
3 marie	6.0 kid
1 john	0.5 infant
8 roger	35.0 adult
8 roger	35.0 middle aged person
2 freddy	2.0 toddler
2 freddy	2.0 kid

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## OutlierFilter (ML Engine)

The OutlierFilter function is useful for filtering a numeric data set before applying ML Engine functions for which outliers can skew the estimates of parameters and cause inaccurate predictions. Such functions include time series functions, GLM, LAR, LinReg, PCA, and KMeans. The input data set is expected to have millions of attribute-value pairs.

The OutlierFilter function filters outliers from a data set, either deleting them or replacing them with a specified value. Optionally, the function stores the outliers in their own table. The function provides these methods for filtering outliers:

- Percentile
- Tukey's test
- Carling's modification to Tukey's test
- Median absolute deviation

The method determines the criteria for an observation to classify as an outlier.

### OutlierFilter Syntax

#### Version 1.10

```
SELECT * FROM OutlierFilter (
  ON { table | view | (query) } AS InputTable
  OUT TABLE OutputTable (output_table)
  [ OUT TABLE OutliersTable (outliers_table) ]
  USING
  TargetColumns ('target_column' [...])
  [ GroupByColumns ('group_column' [...]) ]
  [ OutlierMethod ('method' [...]) ]
  [ ApproxPercentile ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ PercentileThreshold (perc_lower, perc_upper) ]
  [ PercentAccuracy (accuracy) ]
  [ IQRMultiplier (k) ]
  [ RemoveTail ({ 'both' | 'upper' | 'lower' }) ]
  [ ReplacementValue ({ 'delete' | 'null' | 'median' | 'newval' }) ]
  [ MADScaleConstant (constant) ]
  [ MADThreshold (madlimit) ]
) AS alias;
```

## OutlierFilter Syntax Elements

### OutputTable

Specify the name of the table where the function stores the copy of the InputTable (including the PARTITION BY column) with the outliers either deleted (by default) or replaced (as specified by the ReplacementValue syntax element).

### OutliersTable

[Optional] Specify the name of the table where the function outputs copies of the rows of the InputTable that contain outliers.

Default behavior: Function does not output an outlier table.

### TargetColumns

Specify the names of the InputTable columns that contain numeric data to filter.

### GroupByColumns

[Optional] Specify the names of the InputTable columns by which to group the data. If the data schema format is *name:value*, this list must include *name*.

Default behavior: Function does not group data.

### OutlierMethod

[Optional] Specify one or more of the following methods of filtering outliers:

<i>method</i>	Description
'percentile' (Default)	Percentile.
'tukey'	Tukey's test: An outlier is defined as any observation smaller than $V1 - k*(V3-V1)$ or larger than $V3 + k*(V3-V1)$ , where V1 and V3 are 25th and 75th percentiles of data and <i>k</i> is specified by IQRMultiplier syntax element.
'carling'	Carling's modification to Tukey's test: An outlier is defined as an observation outside the range $V2 \pm c*(V3 - V1)$ , where V2 is median of data, V1 and V3 are 25th and 75th percentiles of data, and <i>c</i> is constant (which you cannot change). For more information about Carling's modification, see: <i>Carling, Kenneth. "Resistant outlier rules and the non-Gaussian case." Computational Statistics and Data Analysis 33, no. 3 (2000): 249-258.</i> Available at <a href="https://core.ac.uk/download/pdf/6559387.pdf">https://core.ac.uk/download/pdf/6559387.pdf</a> .
'MAD-median'	Median absolute deviation (MAD), median of absolute values of residuals. For example, for <i>i</i> datapoints and median value of data <i>M</i> , $MAD = \text{median}_i( x_i - M )$ .

Specify either one *method*, which the function uses for all columns specified by TargetColumns, or specify a *method* for each column specified by TargetColumns.

**ApproxPercentile**

[Optional] Specify whether the function calculates the percentiles used as filter limits exactly. Approximate percentiles are typically faster, but can fail when the number of groups exceeds one million.

Default: 'false'

**PercentileThreshold**

[Optional] Specify the range of percentile values for 'percentile' filtering, [*perc\_lower*, 100 - *perc\_lower*].

Default: [5, 95]

**PercentAccuracy**

[Optional] Specify the accuracy of percentiles used for filtering. The value *accuracy* must be in the range [0.01, 50].

Default: 0.5%

**IQRMultiplier**

[Optional] Specify the multiplier of interquartile range for 'tukey' filtering.

Default: 1.5

**RemoveTail**

[Optional] Specify the side of the distribution to filter.

Default: 'both'

**ReplacementValue**

[Optional] Specify how the function handles outliers:

Option	Description
'delete' (Default)	Function does not copy row to output table.
'null'	Function copies row to output table, replacing each outlier with value NULL.
'median'	Function copies row to output table, replacing each outlier with median value for its group.
<i>newval</i>	Function copies row to output table, replacing each outlier with <i>newval</i> , which must be a numeric value.

**MadScaleConstant**

[Optional] Specify the scale constant used with 'MAD-median' filtering; a DOUBLE PRECISION value.

Default: 1.4826, which means  $MAD = 1.4826 * \text{median}(|x - \text{median}(x)|)$

**MadThreshold**

[Optional] Specify the threshold used with 'MAD-median' filtering; a DOUBLE PRECISION value.

Default: 3, which means that  $|x - \text{median}(x)| / MAD > 3$  is flagged as an outlier

## OutlierFilter Input

### InputTable Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, NUMERIC, or DOUBLE PRECISION	[Column appears once for each specified <i>target_column</i> .] Numeric data to filter.
<i>group_column</i>	Any	[Column appears once for each specified <i>group_column</i> .] Column by which to group data.

## OutlierFilter Output

### Output Message Schema

Column	Data Type	Description
message	VARCHAR	Reports whether tables were created successfully.

### OutputTable Schema

The table has the same schema as the [OutlierFilter Input](#) table.

### OutliersTable Schema

This table appears only with OutliersTable syntax element. It has the same schema as the [OutlierFilter Input](#) table.

## OutlierFilter Examples

### OutlierFilter Example: OutlierMethod ('percentile'), ReplacementValue ('null')

#### Input

The InputTable has a time series of atmospheric pressure readings (in mbar) for five cities.

InputTable: ville\_pressuredata

sn	city	period	pressure_mbar
1	Asheville	2010-01-01 00:00:00	1020.5

sn	city	period	pressure_mbar
2	Asheville	2010-01-01 01:00:00	9000
3	Asheville	2010-01-01 02:00:00	1020
4	Asheville	2010-01-01 03:00:00	10000
5	Asheville	2010-01-01 04:00:00	1020.2
6	Asheville	2010-01-01 05:00:00	1020
7	Asheville	2010-01-01 06:00:00	1020.3
8	Asheville	2010-01-01 07:00:00	1020.8
9	Asheville	2010-01-01 08:00:00	1020.3
10	Asheville	2010-01-01 09:00:00	1020.7
...	...	...	...
25	Greenville	2010-01-01 00:00:00	1020.6
26	Greenville	2010-01-01 01:00:00	9000
27	Greenville	2010-01-01 02:00:00	1020.1
28	Greenville	2010-01-01 03:00:00	10000
29	Greenville	2010-01-01 04:00:00	1020.2
30	Greenville	2010-01-01 05:00:00	1020
...	...	...	...
49	Brownsville	2010-01-01 00:00:00	1020.5
50	Brownsville	2010-01-01 01:00:00	9000
51	Brownsville	2010-01-01 02:00:00	1020
52	Brownsville	2010-01-01 03:00:00	10000
53	Brownsville	2010-01-01 04:00:00	1020.2
54	Brownsville	2010-01-01 05:00:00	1020
...	...	...	...
73	Nashville	2010-01-01 00:00:00	1020.4
74	Nashville	2010-01-01 01:00:00	9000
75	Nashville	2010-01-01 02:00:00	1019.9
76	Nashville	2010-01-01 03:00:00	10000
77	Nashville	2010-01-01 04:00:00	1020.1

sn	city	period	pressure_mbar
78	Nashville	2010-01-01 05:00:00	1019.9
...	...	...	...
97	Knoxville	2010-01-01 00:00:00	1020.4
98	Knoxville	2010-01-01 01:00:00	9000
99	Knoxville	2010-01-01 02:00:00	1019.9
100	Knoxville	2010-01-01 03:00:00	10000
101	Knoxville	2010-01-01 04:00:00	1020
102	Knoxville	2010-01-01 05:00:00	1019.9
...	...	...	...

## SQL Call

```
SELECT * FROM OutlierFilter (
  ON ville_pressuredata AS InputTable
  OUT TABLE OutputTable (of_output1)
  USING
    TargetColumns ('pressure_mbar ')
    OutlierMethod ('percentile')
    PercentileThreshold (1,90)
    RemoveTail ('both')
    ReplacementValue ('null')
    GroupByColumns ('city')
) AS dt ;
```

## Output

The outlying values have been replaced by NULL.

message

-----  
Output tables created successfully

```
SELECT * FROM of_output1 ORDER BY 1,2,3;
```

sn	city	period	pressure_mbar
1	ashville	2010-01-01 00:00:00.000000	1020.5
2	ashville	2010-01-01 01:00:00.000000	NULL
3	ashville	2010-01-01 02:00:00.000000	1020.0
4	ashville	2010-01-01 03:00:00.000000	NULL

5	ashville	2010-01-01 04:00:00.000000	1020.2
6	ashville	2010-01-01 05:00:00.000000	1020.0
7	ashville	2010-01-01 06:00:00.000000	1020.3
8	ashville	2010-01-01 07:00:00.000000	1020.8
9	ashville	2010-01-01 08:00:00.000000	1021.3
10	ashville	2010-01-01 09:00:00.000000	1021.7
11	ashville	2010-01-01 10:00:00.000000	1022.1
12	ashville	2010-01-01 11:00:00.000000	1022.0
13	ashville	2010-01-01 12:00:00.000000	1021.1
14	ashville	2010-01-01 13:00:00.000000	1020.0
15	ashville	2010-01-01 14:00:00.000000	1019.3
16	ashville	2010-01-01 15:00:00.000000	1019.0
17	ashville	2010-01-01 16:00:00.000000	1019.2
18	ashville	2010-01-01 17:00:00.000000	1019.6
19	ashville	2010-01-01 18:00:00.000000	1020.1
20	ashville	2010-01-01 19:00:00.000000	1020.6
21	ashville	2010-01-01 20:00:00.000000	1020.9
22	ashville	2010-01-01 21:00:00.000000	1021.1
23	ashville	2010-01-01 22:00:00.000000	1021.0
24	ashville	2010-01-01 23:00:00.000000	1020.9
25	greenville	2010-01-01 00:00:00.000000	1020.6
26	greenville	2010-01-01 01:00:00.000000	NULL
27	greenville	2010-01-01 02:00:00.000000	1020.1
28	greenville	2010-01-01 03:00:00.000000	NULL
29	greenville	2010-01-01 04:00:00.000000	1020.2
30	greenville	2010-01-01 05:00:00.000000	1020.0
31	greenville	2010-01-01 06:00:00.000000	1020.4
32	greenville	2010-01-01 07:00:00.000000	1020.8
33	greenville	2010-01-01 08:00:00.000000	1021.3
34	greenville	2010-01-01 09:00:00.000000	1021.7
35	greenville	2010-01-01 10:00:00.000000	1022.0
36	greenville	2010-01-01 11:00:00.000000	1021.9
37	greenville	2010-01-01 12:00:00.000000	1021.1
38	greenville	2010-01-01 13:00:00.000000	1020.0
39	greenville	2010-01-01 14:00:00.000000	1019.3
40	greenville	2010-01-01 15:00:00.000000	1019.0
41	greenville	2010-01-01 16:00:00.000000	1019.2
42	greenville	2010-01-01 17:00:00.000000	1019.6
43	greenville	2010-01-01 18:00:00.000000	1020.1
44	greenville	2010-01-01 19:00:00.000000	1020.6
45	greenville	2010-01-01 20:00:00.000000	1020.9
46	greenville	2010-01-01 21:00:00.000000	1021.0
47	greenville	2010-01-01 22:00:00.000000	1020.9
48	greenville	2010-01-01 23:00:00.000000	1020.9

49	brownsville	2010-01-01	00:00:00.000000	1020.5
50	brownsville	2010-01-01	01:00:00.000000	NULL
51	brownsville	2010-01-01	02:00:00.000000	1020.0
52	brownsville	2010-01-01	03:00:00.000000	NULL
53	brownsville	2010-01-01	04:00:00.000000	1020.2
54	brownsville	2010-01-01	05:00:00.000000	1020.0
55	brownsville	2010-01-01	06:00:00.000000	1020.3
56	brownsville	2010-01-01	07:00:00.000000	1020.8
57	brownsville	2010-01-01	08:00:00.000000	1021.2
58	brownsville	2010-01-01	09:00:00.000000	1021.6
59	brownsville	2010-01-01	10:00:00.000000	1022.0
60	brownsville	2010-01-01	11:00:00.000000	1021.9
61	brownsville	2010-01-01	12:00:00.000000	1021.0
62	brownsville	2010-01-01	13:00:00.000000	1019.9
63	brownsville	2010-01-01	14:00:00.000000	1019.2
64	brownsville	2010-01-01	15:00:00.000000	1019.0
65	brownsville	2010-01-01	16:00:00.000000	1019.2
66	brownsville	2010-01-01	17:00:00.000000	1019.6
67	brownsville	2010-01-01	18:00:00.000000	1020.0
68	brownsville	2010-01-01	19:00:00.000000	1020.5
69	brownsville	2010-01-01	20:00:00.000000	1020.8
70	brownsville	2010-01-01	21:00:00.000000	1020.9
71	brownsville	2010-01-01	22:00:00.000000	1020.9
72	brownsville	2010-01-01	23:00:00.000000	1020.8
73	nashville	2010-01-01	00:00:00.000000	1020.4
74	nashville	2010-01-01	01:00:00.000000	NULL
75	nashville	2010-01-01	02:00:00.000000	1019.9
76	nashville	2010-01-01	03:00:00.000000	NULL
77	nashville	2010-01-01	04:00:00.000000	1020.1
78	nashville	2010-01-01	05:00:00.000000	1019.9
79	nashville	2010-01-01	06:00:00.000000	1020.2
80	nashville	2010-01-01	07:00:00.000000	1020.6
81	nashville	2010-01-01	08:00:00.000000	1021.1
82	nashville	2010-01-01	09:00:00.000000	1021.5
83	nashville	2010-01-01	10:00:00.000000	1021.9
84	nashville	2010-01-01	11:00:00.000000	1021.8
85	nashville	2010-01-01	12:00:00.000000	1021.0
86	nashville	2010-01-01	13:00:00.000000	1019.8
87	nashville	2010-01-01	14:00:00.000000	1019.2
88	nashville	2010-01-01	15:00:00.000000	1018.9
89	nashville	2010-01-01	16:00:00.000000	1019.1
90	nashville	2010-01-01	17:00:00.000000	1019.5
91	nashville	2010-01-01	18:00:00.000000	1019.9
92	nashville	2010-01-01	19:00:00.000000	1020.4

93	nashville	2010-01-01 20:00:00.000000	1020.7
94	nashville	2010-01-01 21:00:00.000000	1020.9
95	nashville	2010-01-01 22:00:00.000000	1020.8
96	nashville	2010-01-01 23:00:00.000000	1020.7
97	knoxville	2010-01-01 00:00:00.000000	1020.4
98	knoxville	2010-01-01 01:00:00.000000	NULL
99	knoxville	2010-01-01 02:00:00.000000	1019.9
100	knoxville	2010-01-01 03:00:00.000000	NULL
101	knoxville	2010-01-01 04:00:00.000000	1020.0
102	knoxville	2010-01-01 05:00:00.000000	1019.9
103	knoxville	2010-01-01 06:00:00.000000	1020.2
104	knoxville	2010-01-01 07:00:00.000000	1020.6
105	knoxville	2010-01-01 08:00:00.000000	1021.1
106	knoxville	2010-01-01 09:00:00.000000	1021.5
107	knoxville	2010-01-01 10:00:00.000000	1021.9
108	knoxville	2010-01-01 11:00:00.000000	1021.8
109	knoxville	2010-01-01 12:00:00.000000	1021.0
110	knoxville	2010-01-01 13:00:00.000000	1019.9
111	knoxville	2010-01-01 14:00:00.000000	1019.2
112	knoxville	2010-01-01 15:00:00.000000	1018.9
113	knoxville	2010-01-01 16:00:00.000000	1019.2
114	knoxville	2010-01-01 17:00:00.000000	1019.6
115	knoxville	2010-01-01 18:00:00.000000	1020.0
116	knoxville	2010-01-01 19:00:00.000000	1020.5
117	knoxville	2010-01-01 20:00:00.000000	1020.8
118	knoxville	2010-01-01 21:00:00.000000	1020.9
119	knoxville	2010-01-01 22:00:00.000000	1020.9
120	knoxville	2010-01-01 23:00:00.000000	1020.8

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## OutlierFilter Example: OutlierMethod ('MAD-median'), ReplacementValue ('median')

### Input

- InputTable: ville\_pressuredata, as in [OutlierFilter Example: OutlierMethod \('percentile'\), ReplacementValue \('null'\)](#)

### SQL Call

```
SELECT * FROM OutlierFilter (
  ON ville_pressuredata AS InputTable
  OUT TABLE OutputTable (of_output2)
```

```

OUT TABLE OutliersTable (of_outlier2)
USING
  TargetColumns ('pressure_mbar')
  ReplacementValue ('median')
  OutlierMethod ('MAD-median')
  MADScaleConstant (1.4826)
  MADThreshold (3)
  GroupByColumns ('city')
) AS dt ;

```

## Output

The outlying values have been replaced with the median value for the group.

message

-----  
 Output tables created successfully

```
SELECT * FROM of_output2 ORDER BY 1, 2, 3;
```

sn	city	period	pressure_mbar
1	ashville	2010-01-01 00:00:00.000000	1020.5
2	ashville	2010-01-01 01:00:00.000000	1020.8
3	ashville	2010-01-01 02:00:00.000000	1020.0
4	ashville	2010-01-01 03:00:00.000000	1020.8
5	ashville	2010-01-01 04:00:00.000000	1020.2
6	ashville	2010-01-01 05:00:00.000000	1020.0
7	ashville	2010-01-01 06:00:00.000000	1020.3
8	ashville	2010-01-01 07:00:00.000000	1020.8
9	ashville	2010-01-01 08:00:00.000000	1021.3
10	ashville	2010-01-01 09:00:00.000000	1021.7
11	ashville	2010-01-01 10:00:00.000000	1022.1
12	ashville	2010-01-01 11:00:00.000000	1022.0
13	ashville	2010-01-01 12:00:00.000000	1021.1
14	ashville	2010-01-01 13:00:00.000000	1020.0
15	ashville	2010-01-01 14:00:00.000000	1019.3
16	ashville	2010-01-01 15:00:00.000000	1019.0
17	ashville	2010-01-01 16:00:00.000000	1019.2
18	ashville	2010-01-01 17:00:00.000000	1019.6
19	ashville	2010-01-01 18:00:00.000000	1020.1
20	ashville	2010-01-01 19:00:00.000000	1020.6
21	ashville	2010-01-01 20:00:00.000000	1020.9
22	ashville	2010-01-01 21:00:00.000000	1021.1
23	ashville	2010-01-01 22:00:00.000000	1021.0
24	ashville	2010-01-01 23:00:00.000000	1020.9

25	greenville	2010-01-01	00:00:00.000000	1020.6
26	greenville	2010-01-01	01:00:00.000000	1020.8
27	greenville	2010-01-01	02:00:00.000000	1020.1
28	greenville	2010-01-01	03:00:00.000000	1020.8
29	greenville	2010-01-01	04:00:00.000000	1020.2
30	greenville	2010-01-01	05:00:00.000000	1020.0
31	greenville	2010-01-01	06:00:00.000000	1020.4
32	greenville	2010-01-01	07:00:00.000000	1020.8
33	greenville	2010-01-01	08:00:00.000000	1021.3
34	greenville	2010-01-01	09:00:00.000000	1021.7
35	greenville	2010-01-01	10:00:00.000000	1022.0
36	greenville	2010-01-01	11:00:00.000000	1021.9
37	greenville	2010-01-01	12:00:00.000000	1021.1
38	greenville	2010-01-01	13:00:00.000000	1020.0
39	greenville	2010-01-01	14:00:00.000000	1019.3
40	greenville	2010-01-01	15:00:00.000000	1019.0
41	greenville	2010-01-01	16:00:00.000000	1019.2
42	greenville	2010-01-01	17:00:00.000000	1019.6
43	greenville	2010-01-01	18:00:00.000000	1020.1
44	greenville	2010-01-01	19:00:00.000000	1020.6
45	greenville	2010-01-01	20:00:00.000000	1020.9
46	greenville	2010-01-01	21:00:00.000000	1021.0
47	greenville	2010-01-01	22:00:00.000000	1020.9
48	greenville	2010-01-01	23:00:00.000000	1020.9
49	brownsville	2010-01-01	00:00:00.000000	1020.5
50	brownsville	2010-01-01	01:00:00.000000	1020.8
51	brownsville	2010-01-01	02:00:00.000000	1020.0
52	brownsville	2010-01-01	03:00:00.000000	1020.8
53	brownsville	2010-01-01	04:00:00.000000	1020.2
54	brownsville	2010-01-01	05:00:00.000000	1020.0
55	brownsville	2010-01-01	06:00:00.000000	1020.3
56	brownsville	2010-01-01	07:00:00.000000	1020.8
57	brownsville	2010-01-01	08:00:00.000000	1021.2
58	brownsville	2010-01-01	09:00:00.000000	1021.6
59	brownsville	2010-01-01	10:00:00.000000	1022.0
60	brownsville	2010-01-01	11:00:00.000000	1021.9
61	brownsville	2010-01-01	12:00:00.000000	1021.0
62	brownsville	2010-01-01	13:00:00.000000	1019.9
63	brownsville	2010-01-01	14:00:00.000000	1019.2
64	brownsville	2010-01-01	15:00:00.000000	1019.0
65	brownsville	2010-01-01	16:00:00.000000	1019.2
66	brownsville	2010-01-01	17:00:00.000000	1019.6
67	brownsville	2010-01-01	18:00:00.000000	1020.0
68	brownsville	2010-01-01	19:00:00.000000	1020.5

69	brownsville	2010-01-01	20:00:00.000000	1020.8
70	brownsville	2010-01-01	21:00:00.000000	1020.9
71	brownsville	2010-01-01	22:00:00.000000	1020.9
72	brownsville	2010-01-01	23:00:00.000000	1020.8
73	nashville	2010-01-01	00:00:00.000000	1020.4
74	nashville	2010-01-01	01:00:00.000000	1020.6
75	nashville	2010-01-01	02:00:00.000000	1019.9
76	nashville	2010-01-01	03:00:00.000000	1020.6
77	nashville	2010-01-01	04:00:00.000000	1020.1
78	nashville	2010-01-01	05:00:00.000000	1019.9
79	nashville	2010-01-01	06:00:00.000000	1020.2
80	nashville	2010-01-01	07:00:00.000000	1020.6
81	nashville	2010-01-01	08:00:00.000000	1021.1
82	nashville	2010-01-01	09:00:00.000000	1021.5
83	nashville	2010-01-01	10:00:00.000000	1021.9
84	nashville	2010-01-01	11:00:00.000000	1021.8
85	nashville	2010-01-01	12:00:00.000000	1021.0
86	nashville	2010-01-01	13:00:00.000000	1019.8
87	nashville	2010-01-01	14:00:00.000000	1019.2
88	nashville	2010-01-01	15:00:00.000000	1018.9
89	nashville	2010-01-01	16:00:00.000000	1019.1
90	nashville	2010-01-01	17:00:00.000000	1019.5
91	nashville	2010-01-01	18:00:00.000000	1019.9
92	nashville	2010-01-01	19:00:00.000000	1020.4
93	nashville	2010-01-01	20:00:00.000000	1020.7
94	nashville	2010-01-01	21:00:00.000000	1020.9
95	nashville	2010-01-01	22:00:00.000000	1020.8
96	nashville	2010-01-01	23:00:00.000000	1020.7
97	knoxville	2010-01-01	00:00:00.000000	1020.4
98	knoxville	2010-01-01	01:00:00.000000	1020.6
99	knoxville	2010-01-01	02:00:00.000000	1019.9
100	knoxville	2010-01-01	03:00:00.000000	1020.6
101	knoxville	2010-01-01	04:00:00.000000	1020.0
102	knoxville	2010-01-01	05:00:00.000000	1019.9
103	knoxville	2010-01-01	06:00:00.000000	1020.2
104	knoxville	2010-01-01	07:00:00.000000	1020.6
105	knoxville	2010-01-01	08:00:00.000000	1021.1
106	knoxville	2010-01-01	09:00:00.000000	1021.5
107	knoxville	2010-01-01	10:00:00.000000	1021.9
108	knoxville	2010-01-01	11:00:00.000000	1021.8
109	knoxville	2010-01-01	12:00:00.000000	1021.0
110	knoxville	2010-01-01	13:00:00.000000	1019.9
111	knoxville	2010-01-01	14:00:00.000000	1019.2
112	knoxville	2010-01-01	15:00:00.000000	1018.9

113	knoxville	2010-01-01 16:00:00.000000	1019.2
114	knoxville	2010-01-01 17:00:00.000000	1019.6
115	knoxville	2010-01-01 18:00:00.000000	1020.0
116	knoxville	2010-01-01 19:00:00.000000	1020.5
117	knoxville	2010-01-01 20:00:00.000000	1020.8
118	knoxville	2010-01-01 21:00:00.000000	1020.9
119	knoxville	2010-01-01 22:00:00.000000	1020.9
120	knoxville	2010-01-01 23:00:00.000000	1020.8

```
SELECT * FROM of_outlier2 ORDER BY 1, 2, 3;
```

sn	city	period	pressure_mbar
2	ashville	2010-01-01 01:00:00.000000	9000.0
4	ashville	2010-01-01 03:00:00.000000	10000.0
26	greenville	2010-01-01 01:00:00.000000	9000.0
28	greenville	2010-01-01 03:00:00.000000	10000.0
50	brownsville	2010-01-01 01:00:00.000000	9000.0
52	brownsville	2010-01-01 03:00:00.000000	10000.0
74	nashville	2010-01-01 01:00:00.000000	9000.0
76	nashville	2010-01-01 03:00:00.000000	10000.0
98	knoxville	2010-01-01 01:00:00.000000	9000.0
100	knoxville	2010-01-01 03:00:00.000000	10000.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pack\_MLE (ML Engine)

The Pack\_MLE function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack\_MLE complements the function [Unpack\\_MLE \(ML Engine\)](#), but you can use it on any columns that meet the input requirements.

---

### Note:

To use Pack\_MLE and Unpack\_MLE together, you must run both on ML Engine platform. Pack\_MLE and Unpack\_MLE are incompatible with Advanced SQL Engine Pack and Unpack functions.

---

Before packing columns, note their data types—you need them if you want to unpack the packed column.

## Pack\_MLE Syntax

### Version 1.6

```
SELECT * FROM Pack_MLE (
  ON { table | view | (query) }
  USING
  [ TargetColumns ({ 'target_column' | target_column_range }[,... ]) ]
  [ Delimiter ('delimiter') ]
  [ IncludeColumnName ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  OutputColumn ('output_column')
) AS alias;
```

### Related Information:

[Column Specification Syntax Elements](#)

## Pack\_MLE Syntax Elements

### TargetColumns

[Optional] Specify the names of the input columns to pack into a single output column. These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

### Delimiter

[Optional] Specify the delimiter (a string) that separates the virtual columns in the packed data.

Default: ',' (comma)

#### IncludeColumnName

[Optional] Specify whether to label each virtual column value with its column name (making the virtual column *target\_column:value*).

Default: 'true'

#### OutputColumn

Specify the name to give to the packed output column.

## Pack\_MLE Input

### Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Column to pack, with other target columns, into single output column.
<i>other_input_column</i>	Any	[Column appears zero or more times.] Column to copy to output table.

## Pack\_MLE Output

### Output Table Schema

Column	Data Type	Description
<i>row_id</i>	BIGINT	Column created by function. Value may vary from run to run on same data set.
<i>output_column</i>	CLOB	Packed column.
<i>other_input_column</i>	Same as in input table	[Column appears once for each specified <i>other_input_column</i> .] Column copied from input table.

## Pack\_MLE Examples

### Pack\_MLE Example: Default Values

#### Input

The input table, *ville\_temperature*, contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville\_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2010-01-01 00:00:00	35.1
2	Nashville	Tennessee	2010-01-01 01:00:00	36.2
3	Nashville	Tennessee	2010-01-01 02:00:00	34.5
4	Nashville	Tennessee	2010-01-01 03:00:00	33.6
5	Nashville	Tennessee	2010-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2010-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2010-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2010-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2010-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2010-01-01 07:00:00	32.4

## SQL Call

Delimiter and IncludeColumnName have their default values.

```
SELECT row_id, cast(packed_data as varchar(100)), sn
  FROM Pack_MLE (
    ON ville_temperature
    USING
    Delimiter(',')
    OutputColumn('packed_data')
    IncludeColumnName('true')
    TargetColumns('city', 'state', 'period', 'temp_F')
  ) AS dt ORDER BY sn;
```

## Output

The columns specified by TargetColumns are packed in the column packed\_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

row_id	packed_data	sn
3	city:nashville,state:tennessee,period:2010-01-01 00:00:00,temp_f:35.1	1
5	city:nashville,state:tennessee,period:2010-01-01 01:00:00,temp_f:36.2	2
2	city:nashville,state:tennessee,period:2010-01-01 02:00:00,temp_f:34.5	3
2	city:nashville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.6	4
1	city:nashville,state:tennessee,period:2010-01-01 04:00:00,temp_f:33.1	5

```

3 city:knoxville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.2 6
1 city:knoxville,state:tennessee,period:2010-01-01 04:00:00,temp_f:32.8 7
4 city:knoxville,state:tennessee,period:2010-01-01 05:00:00,temp_f:32.4 8
6 city:knoxville,state:tennessee,period:2010-01-01 06:00:00,temp_f:32.2 9
7 city:knoxville,state:tennessee,period:2010-01-01 07:00:00,temp_f:32.4 10

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pack\_MLE Example: Nondefault Values

### Input

- Input table: ville\_temperature, as in [Pack\\_MLE Example: Default Values](#)

### SQL Call

Delimiter and IncludeColumnName have nondefault values.

```

SELECT row_id, cast(packed_data as varchar(100)), sn
FROM Pack_MLE(
  ON ville_temperature
  USING
  Delimiter('|')
  OutputColumn('packed_data')
  IncludeColumnName('false')
  TargetColumns('city', 'state', 'period', 'temp_F')
) as dt ORDER BY sn;

```

### Output

Virtual columns are separated by pipe characters and not labeled with their column names.

row_id	packed_data	sn
5	nashville tennessee 2010-01-01 00:00:00 35.1	1
7	nashville tennessee 2010-01-01 01:00:00 36.2	2
4	nashville tennessee 2010-01-01 02:00:00 34.5	3
2	nashville tennessee 2010-01-01 03:00:00 33.6	4
3	nashville tennessee 2010-01-01 04:00:00 33.1	5
3	knoxville tennessee 2010-01-01 03:00:00 33.2	6
1	knoxville tennessee 2010-01-01 04:00:00 32.8	7
6	knoxville tennessee 2010-01-01 05:00:00 32.4	8
1	knoxville tennessee 2010-01-01 06:00:00 32.2	9
2	knoxville tennessee 2010-01-01 07:00:00 32.4	10

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pivoting (ML Engine)

The Pivoting function pivots data that is stored in rows into columns. It outputs a table whose columns are based on the individual values from an input table column. The output table schema depends on the function syntax elements. The function handles missing or NULL values automatically.

The reverse of this function is [Unpivoting \(ML Engine\)](#).

### Pivoting Syntax

#### Version 1.9

```
SELECT * FROM Pivoting (
  ON { table | view | (query) } PARTITION BY partition_column [,...]
  [ ORDER BY order_column ]
  USING
  PartitionColumns ({ 'partition_column' | partition_column_range }[,...])
  { NumberOfRows (number_of_rows) |
    PivotColumn ('pivot_column')
    [ PivotKeys ('pivot_key' [,...]) ]
    [ NumericPivotKey ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]
  }
  TargetColumns ({ 'target_column' | 'target_column_range' }[,...])
) AS alias;
```

#### Related Information:

[Column Specification Syntax Elements](#)

### Pivoting Syntax Elements

#### PartitionColumns

Specify the same columns as the PARTITION BY clause (in any order).

#### NumberOfRows

[Required if you omit PivotColumn.] Use NumberOfRows when no column contains pivot keys, but you can specify a maximum number of rows in any partition. The function pivots the input rows into this number of columns in the output table.

If a partition has fewer than *number\_of\_rows* rows, the function adds NULL values; if a partition has more than *number\_of\_rows* rows, the function omits the extra rows.

If you use NumberOfRows, you must use the ORDER BY clause to order the input data; otherwise, the contents of the output table columns may vary from run to run.

**PivotColumn**

[Required if you omit NumberOfRows.] Specify the name of the input column that contains the pivot keys.

If *pivot\_column* contains numeric values, the function casts them to VARCHAR; function performance improves slightly if you specify NumericPivotKey ('true').

**PivotKeys**

[Required if you specify PivotColumn.] Specify the values in *pivot\_column* to use as pivot keys. The function ignores rows that contain other values in *pivot\_column*.

**NumericPivotKey**

[Optional] Use this syntax element only with the PivotColumn syntax element. If *pivot\_column* is numeric, NumericPivotKey ('true') improves function performance slightly.

Default: 'false'

**TargetColumns**

[Optional] Specify the names of the target columns (input columns that contain the values to pivot).

## Pivoting Input

**Input Table Schema**

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which to partition input data.
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Values to pivot.

## Pivoting Output

The output table schema depends on whether you specify the syntax element NumberOfRows or PivotColumn.

**Output Table Schema, NumberOfRows**

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned.
<i>value_i</i>	Any	[Column appears <i>number_of_rows</i> times.] Value in <i>i</i> th target column, where <i>i</i> is in range [0, <i>number_of_rows</i> -1]. Columns appear in order specified by ORDER BY clause.

**Output Table Schema, PivotColumn**

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned.
<i>value_target_column</i>	Any	[Column appears once for each <i>pivot_key</i> .] Values for <i>pivot_key</i> that are associated with partitions in row.

## Pivoting Examples

### Pivoting Example: NumberOfRows

This example specifies the NumberOfRows syntax element.

#### Input

The input table, `pivot_input`, contains temperature, pressure, and dewpoint data for three cities, in sparse format.

**pivot\_input**

sn	city	week	attribute	value1
1	Asheville	1	temp	32
1	Asheville	1	pressure	1020.8
1	Asheville	1	dewpoint	27.6F
2	Asheville	2	temp	32
2	Asheville	2	pressure	1021.3
2	Asheville	2	dewpoint	27.4F
3	Asheville	3	temp	34
3	Asheville	3	pressure	1021.7
3	Asheville	3	dewpoint	28.2F
4	Nashville	1	temp	42
4	Nashville	1	pressure	1021
4	Nashville	1	dewpoint	29.4F
5	Nashville	2	temp	44
5	Nashville	2	pressure	1019.8

sn	city	week	attribute	value1
5	Nashville	2	dewpoint	29.2F
6	Brownsville	2	temp	47
6	Brownsville	2	pressure	1019
6	Brownsville	2	dewpoint	28.9F
7	Brownsville	3	temp	46
7	Brownsville	3	pressure	1019.2
7	Brownsville	3	dewpoint	28.9F

## SQL Call

```
SELECT * FROM Pivoting (
  ON pivot_input PARTITION BY sn,city,week
  ORDER BY week,attribute
  USING
  PartitionColumns ('sn','city', 'week')
  NumberOfRows (3)
  TargetColumns ('value1')
) AS dt ORDER BY 1,2,3;
```

The ORDER BY clause is required. If omitted, the output table column content is nondeterministic (for more information, see [Nondeterministic Results and UniqueID Syntax Element](#)). The function adds any NULL values at the end.

## Output

The function outputs the input column contents in dense format in the output columns value1\_0, value1\_1, and value1\_2, which contain the dewpoint, pressure, and temperature, respectively. Because these values are numeric, the function casts them to VARCHAR.

sn	city	week	value1_0	value1_1	value1_2
1	asheville	1	27.6f	1020.8	32
2	asheville	2	27.4f	1021.3	32
3	asheville	3	28.2f	1021.7	34
4	nashville	1	29.4f	1021	42
5	nashville	2	29.2f	1019.8	44
6	brownsville	2	28.9f	1019	47
7	brownsville	3	28.9f	1019.2	46

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pivoting Example: PivotKeys

### Input

- Input table: pivot\_input, as in [Pivoting Example: NumberOfRows](#)

### SQL Call

```
SELECT * FROM Pivoting (
  ON pivot_input PARTITION BY sn,city,week
  USING
  PartitionColumns ('sn','city', 'week')
  PivotKeys ('temp','pressure')
  PivotColumn ('attribute')
  TargetColumns ('value1')
) AS dt ORDER BY 1,2,3;
```

With PivotKeys, the function does not use the ORDER BY clause.

### Output

To create the output table, the function pivots the input table on the partition columns (sn, city, and week) and outputs the contents of the target column (value1) in dense format in the output columns value1\_pressure and value1\_temp.

sn	city	week	value1_pressure	value1_temp
1	asheville	1	1020.8	32
2	asheville	2	1021.3	32
3	asheville	3	1021.7	34
4	nashville	1	1021	42
5	nashville	2	1019.8	44
6	brownsville	2	1019	47
7	brownsville	3	1019.2	46

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pack\_MLE (ML Engine)

The Pack\_MLE function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack\_MLE complements the function [Unpack\\_MLE \(ML Engine\)](#), but you can use it on any columns that meet the input requirements.

---

### Note:

To use Pack\_MLE and Unpack\_MLE together, you must run both on ML Engine platform. Pack\_MLE and Unpack\_MLE are incompatible with Advanced SQL Engine Pack and Unpack functions.

---

Before packing columns, note their data types—you need them if you want to unpack the packed column.

## Pack\_MLE Syntax

### Version 1.6

```
SELECT * FROM Pack_MLE (
  ON { table | view | (query) }
  USING
  [ TargetColumns ({ 'target_column' | target_column_range }[,... ]) ]
  [ Delimiter ('delimiter') ]
  [ IncludeColumnName ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  OutputColumn ('output_column')
) AS alias;
```

### Related Information:

[Column Specification Syntax Elements](#)

## Pack\_MLE Syntax Elements

### TargetColumns

[Optional] Specify the names of the input columns to pack into a single output column. These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

### Delimiter

[Optional] Specify the delimiter (a string) that separates the virtual columns in the packed data.

Default: ',' (comma)

#### IncludeColumnName

[Optional] Specify whether to label each virtual column value with its column name (making the virtual column *target\_column:value*).

Default: 'true'

#### OutputColumn

Specify the name to give to the packed output column.

## Pack\_MLE Input

### Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Column to pack, with other target columns, into single output column.
<i>other_input_column</i>	Any	[Column appears zero or more times.] Column to copy to output table.

## Pack\_MLE Output

### Output Table Schema

Column	Data Type	Description
<i>row_id</i>	BIGINT	Column created by function. Value may vary from run to run on same data set.
<i>output_column</i>	CLOB	Packed column.
<i>other_input_column</i>	Same as in input table	[Column appears once for each specified <i>other_input_column</i> .] Column copied from input table.

## Pack\_MLE Examples

### Pack\_MLE Example: Default Values

#### Input

The input table, *ville\_temperature*, contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville\_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2010-01-01 00:00:00	35.1
2	Nashville	Tennessee	2010-01-01 01:00:00	36.2
3	Nashville	Tennessee	2010-01-01 02:00:00	34.5
4	Nashville	Tennessee	2010-01-01 03:00:00	33.6
5	Nashville	Tennessee	2010-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2010-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2010-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2010-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2010-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2010-01-01 07:00:00	32.4

## SQL Call

Delimiter and IncludeColumnName have their default values.

```
SELECT row_id, cast(packed_data as varchar(100)), sn
  FROM Pack_MLE (
    ON ville_temperature
    USING
    Delimiter(',')
    OutputColumn('packed_data')
    IncludeColumnName('true')
    TargetColumns('city', 'state', 'period', 'temp_F')
  ) AS dt ORDER BY sn;
```

## Output

The columns specified by TargetColumns are packed in the column packed\_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

row_id	packed_data	sn
3	city:nashville,state:tennessee,period:2010-01-01 00:00:00,temp_f:35.1	1
5	city:nashville,state:tennessee,period:2010-01-01 01:00:00,temp_f:36.2	2
2	city:nashville,state:tennessee,period:2010-01-01 02:00:00,temp_f:34.5	3
2	city:nashville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.6	4
1	city:nashville,state:tennessee,period:2010-01-01 04:00:00,temp_f:33.1	5

```

3 city:knoxville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.2 6
1 city:knoxville,state:tennessee,period:2010-01-01 04:00:00,temp_f:32.8 7
4 city:knoxville,state:tennessee,period:2010-01-01 05:00:00,temp_f:32.4 8
6 city:knoxville,state:tennessee,period:2010-01-01 06:00:00,temp_f:32.2 9
7 city:knoxville,state:tennessee,period:2010-01-01 07:00:00,temp_f:32.4 10

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pack\_MLE Example: Nondefault Values

### Input

- Input table: ville\_temperature, as in [Pack\\_MLE Example: Default Values](#)

### SQL Call

Delimiter and IncludeColumnName have nondefault values.

```

SELECT row_id, cast(packed_data as varchar(100)), sn
FROM Pack_MLE(
  ON ville_temperature
  USING
  Delimiter('|')
  OutputColumn('packed_data')
  IncludeColumnName('false')
  TargetColumns('city', 'state', 'period', 'temp_F')
) as dt ORDER BY sn;

```

### Output

Virtual columns are separated by pipe characters and not labeled with their column names.

row_id	packed_data	sn
5	nashville tennessee 2010-01-01 00:00:00 35.1	1
7	nashville tennessee 2010-01-01 01:00:00 36.2	2
4	nashville tennessee 2010-01-01 02:00:00 34.5	3
2	nashville tennessee 2010-01-01 03:00:00 33.6	4
3	nashville tennessee 2010-01-01 04:00:00 33.1	5
3	knoxville tennessee 2010-01-01 03:00:00 33.2	6
1	knoxville tennessee 2010-01-01 04:00:00 32.8	7
6	knoxville tennessee 2010-01-01 05:00:00 32.4	8
1	knoxville tennessee 2010-01-01 06:00:00 32.2	9
2	knoxville tennessee 2010-01-01 07:00:00 32.4	10

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

# Teradata Vantage NewSQL Engine Analytic Functions

## Antiselect

Antiselect returns all columns *except* those specified in the Exclude syntax element.

### Note:

This function requires the UTF8 client character set.

## Antiselect Syntax

```
SELECT * FROM Antiselect (
  ON { table | view | (query) }
  USING
  Exclude ({ 'exclude_column' | exclude_column_range }[,...])
) AS alias;
```

## Antiselect Syntax Elements

### Exclude

Specify the names of the input table columns to exclude from the output table. Column names must be valid object names, which are defined in *Teradata Vantage™ SQL Fundamentals*, B035-1141.

The *exclude\_column* is a column name. This is the syntax of *exclude\_column\_range*:

```
'start_column:end_column' [, '-exclude_in-range_column' ]
```

The range includes its endpoints.

The *start\_column* and *end\_column* can be:

- Column names (for example, 'column1:column2')
- Column names must contain only letters in the English alphabet, digits, and special characters. If a column name includes any special characters, surround the column name with double quotation marks. For example, if the column name is a\*b, specify it as "a\*b". A column name cannot contain a double quotation mark.
- Nonnegative integers that represent the indexes of columns in the table (for example, '[0:4]')
- The first column has index 0; therefore, '[0:4]' specifies the first five columns in the table.
- Empty. For example:

- '[:4]' specifies all columns up to and including the column with index 4.
- '[4:]' specifies the column with index 4 and all columns after it.
- '[:]' specifies all columns in the table.

The *exclude\_in-range\_column* is a column in the specified range, represented by either its name or its index (for example, '[0:99]', '-[50]', '-column10' specifies the columns with indexes 0 through 99, except the column with index 50 and column10).

Column ranges cannot overlap, and cannot include any specified *exclude\_column*.

## Antiselect Input

The input table can have any schema.

## Antiselect Output

The output table has all input table columns except those specified by the Exclude syntax element.

## Antiselect Examples

### Antiselect Example: No Column Ranges

#### Input

The input table, *antiselect\_test*, is a sample set of sales data containing 13 columns.

**antiselect\_test**

sno	id	orderdate	priority	qty	sales	disct	dmode	custname	province	region	cust
1	3	2010-10-13 00:00:00	Low	6	261. 54	0.04	Regular Air	Muhammed MacIntyre	Nunavut	Nunavut	Sma Busi
49	293	2012-10-01 00:00:00	High	49	10123	0.07	Delivery Truck	Barry French	Nunavut	Nunavut	Cons
50	293	2012-10-01 00:00:00	High	27	244. 57	0.01	Regular Air	Barry French	Nunavut	Nunavut	Cons
80	483	2011-07-10 00:00:00	High	30	4965. 76	0.08	Regular Air	Clay Rozendal	Nunavut	Nunavut	Corp
85	515	2010-08-28 00:00:00	Not specified	19	394. 27	0.08	Regular Air	Carlos Soltero	Nunavut	Nunavut	Cons
86	515	2010-08-28 00:00:00	Not specified	21	146. 69	0.05	Regular Air	Carlos Soltero	Nunavut	Nunavut	Cons
97	613	2011-06-17 00:00:00	High	12	93.54	0.03	Regular Air	Carl Jackson	Nunavut	Nunavut	Corp

**SQL Call**

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', 'orderdate', 'disct', 'province', 'custsegment')
) AS dt ORDER BY 1, 4;
```

**Output**

sno	priority	qty	sales	dmode	custname	region	prodcats
1	Low	6	2. 61540000000000E 002	Regular Air	Muhammed MacIntyre	Nunavut	Office Supplies
49	High	49	1. 01230000000000E 004	Delivery Truck	Barry French	Nunavut	Office Supplies
50	High	27	2. 44570000000000E 002	Regular Air	Barry French	Nunavut	Office Supplies
80	High	30	4. 96576000000000E 003	Regular Air	Clay Rozendal	Nunavut	Technology
85	Not specified	19	3. 94270000000000E 002	Regular Air	Carlos Soltero	Nunavut	Office Supplies
86	Not specified	21	1. 46690000000000E 002	Regular Air	Carlos Soltero	Nunavut	Furniture
97	High	12	9. 35400000000000E 001	Regular Air	Carl Jackson	Nunavut	Office Supplies

**Antiselect Example: Column Range****Input**

The input table is antiselect\_test, as in [Antiselect Example: No Column Ranges](#).

**SQL Call**

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
```

```
Exclude ('id', '[2:3]', 'custname:prodcat')
) AS dt ORDER BY 1, 4;
```

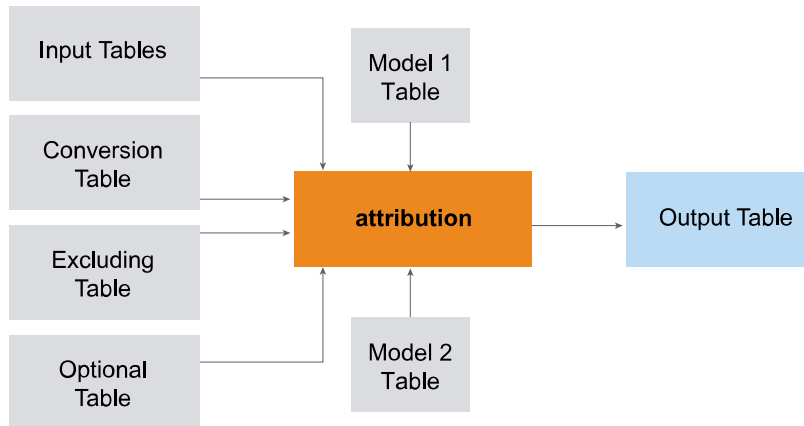
## Output

sno	qty	sales	disct	dmode
1	6	2.61540000000000E 002	0.04	Regular Air
49	49	1.01230000000000E 004	0.07	Delivery Truck
50	27	2.44570000000000E 002	0.01	Regular Air
80	30	4.96576000000000E 003	0.08	Regular Air
85	19	3.94270000000000E 002	0.08	Regular Air
86	21	1.46690000000000E 002	0.05	Regular Air
97	12	9.35400000000000E 001	0.03	Regular Air

## Attribution

The Attribution function is used in web page analysis, where it lets companies assign weights to pages before certain events, such as buying a product.

The function takes data and parameters from multiple tables and outputs attributions.



## Pack\_MLE (ML Engine)

The Pack\_MLE function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack\_MLE complements the function [Unpack\\_MLE \(ML Engine\)](#), but you can use it on any columns that meet the input requirements.

---

### Note:

To use Pack\_MLE and Unpack\_MLE together, you must run both on ML Engine platform. Pack\_MLE and Unpack\_MLE are incompatible with Advanced SQL Engine Pack and Unpack functions.

---

Before packing columns, note their data types—you need them if you want to unpack the packed column.

## Pack\_MLE Syntax

### Version 1.6

```
SELECT * FROM Pack_MLE (
  ON { table | view | (query) }
  USING
  [ TargetColumns ({ 'target_column' | target_column_range }[,... ]) ]
  [ Delimiter ('delimiter') ]
  [ IncludeColumnName ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  OutputColumn ('output_column')
) AS alias;
```

### Related Information:

[Column Specification Syntax Elements](#)

## Pack\_MLE Syntax Elements

### TargetColumns

[Optional] Specify the names of the input columns to pack into a single output column. These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

### Delimiter

[Optional] Specify the delimiter (a string) that separates the virtual columns in the packed data.

Default: ',' (comma)

#### IncludeColumnName

[Optional] Specify whether to label each virtual column value with its column name (making the virtual column *target\_column:value*).

Default: 'true'

#### OutputColumn

Specify the name to give to the packed output column.

## Pack\_MLE Input

### Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Column to pack, with other target columns, into single output column.
<i>other_input_column</i>	Any	[Column appears zero or more times.] Column to copy to output table.

## Pack\_MLE Output

### Output Table Schema

Column	Data Type	Description
<i>row_id</i>	BIGINT	Column created by function. Value may vary from run to run on same data set.
<i>output_column</i>	CLOB	Packed column.
<i>other_input_column</i>	Same as in input table	[Column appears once for each specified <i>other_input_column</i> .] Column copied from input table.

## Pack\_MLE Examples

### Pack\_MLE Example: Default Values

#### Input

The input table, *ville\_temperature*, contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville\_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2010-01-01 00:00:00	35.1
2	Nashville	Tennessee	2010-01-01 01:00:00	36.2
3	Nashville	Tennessee	2010-01-01 02:00:00	34.5
4	Nashville	Tennessee	2010-01-01 03:00:00	33.6
5	Nashville	Tennessee	2010-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2010-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2010-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2010-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2010-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2010-01-01 07:00:00	32.4

## SQL Call

Delimiter and IncludeColumnName have their default values.

```
SELECT row_id, cast(packed_data as varchar(100)), sn
  FROM Pack_MLE (
    ON ville_temperature
    USING
    Delimiter(',')
    OutputColumn('packed_data')
    IncludeColumnName('true')
    TargetColumns('city', 'state', 'period', 'temp_F')
  ) AS dt ORDER BY sn;
```

## Output

The columns specified by TargetColumns are packed in the column packed\_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

row_id	packed_data	sn
3	city:nashville,state:tennessee,period:2010-01-01 00:00:00,temp_f:35.1	1
5	city:nashville,state:tennessee,period:2010-01-01 01:00:00,temp_f:36.2	2
2	city:nashville,state:tennessee,period:2010-01-01 02:00:00,temp_f:34.5	3
2	city:nashville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.6	4
1	city:nashville,state:tennessee,period:2010-01-01 04:00:00,temp_f:33.1	5

```

3 city:knoxville,state:tennessee,period:2010-01-01 03:00:00,temp_f:33.2 6
1 city:knoxville,state:tennessee,period:2010-01-01 04:00:00,temp_f:32.8 7
4 city:knoxville,state:tennessee,period:2010-01-01 05:00:00,temp_f:32.4 8
6 city:knoxville,state:tennessee,period:2010-01-01 06:00:00,temp_f:32.2 9
7 city:knoxville,state:tennessee,period:2010-01-01 07:00:00,temp_f:32.4 10

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pack\_MLE Example: Nondefault Values

### Input

- Input table: ville\_temperature, as in [Pack\\_MLE Example: Default Values](#)

### SQL Call

Delimiter and IncludeColumnName have nondefault values.

```

SELECT row_id, cast(packed_data as varchar(100)), sn
FROM Pack_MLE(
  ON ville_temperature
  USING
  Delimiter('|')
  OutputColumn('packed_data')
  IncludeColumnName('false')
  TargetColumns('city', 'state', 'period', 'temp_F')
) as dt ORDER BY sn;

```

### Output

Virtual columns are separated by pipe characters and not labeled with their column names.

row_id	packed_data	sn
5	nashville tennessee 2010-01-01 00:00:00 35.1	1
7	nashville tennessee 2010-01-01 01:00:00 36.2	2
4	nashville tennessee 2010-01-01 02:00:00 34.5	3
2	nashville tennessee 2010-01-01 03:00:00 33.6	4
3	nashville tennessee 2010-01-01 04:00:00 33.1	5
3	knoxville tennessee 2010-01-01 03:00:00 33.2	6
1	knoxville tennessee 2010-01-01 04:00:00 32.8	7
6	knoxville tennessee 2010-01-01 05:00:00 32.4	8
1	knoxville tennessee 2010-01-01 06:00:00 32.2	9
2	knoxville tennessee 2010-01-01 07:00:00 32.4	10

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pivoting (ML Engine)

The Pivoting function pivots data that is stored in rows into columns. It outputs a table whose columns are based on the individual values from an input table column. The output table schema depends on the function syntax elements. The function handles missing or NULL values automatically.

The reverse of this function is [Unpivoting \(ML Engine\)](#).

### Pivoting Syntax

#### Version 1.9

```
SELECT * FROM Pivoting (
  ON { table | view | (query) } PARTITION BY partition_column [,...]
  [ ORDER BY order_column ]
  USING
  PartitionColumns ({ 'partition_column' | partition_column_range }[,...])
  { NumberOfRows (number_of_rows) |
    PivotColumn ('pivot_column')
    [ PivotKeys ('pivot_key' [,...]) ]
    [ NumericPivotKey ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]
  }
  TargetColumns ({ 'target_column' | 'target_column_range' }[,...])
) AS alias;
```

#### Related Information:

[Column Specification Syntax Elements](#)

### Pivoting Syntax Elements

#### PartitionColumns

Specify the same columns as the PARTITION BY clause (in any order).

#### NumberOfRows

[Required if you omit PivotColumn.] Use NumberOfRows when no column contains pivot keys, but you can specify a maximum number of rows in any partition. The function pivots the input rows into this number of columns in the output table.

If a partition has fewer than *number\_of\_rows* rows, the function adds NULL values; if a partition has more than *number\_of\_rows* rows, the function omits the extra rows.

If you use NumberOfRows, you must use the ORDER BY clause to order the input data; otherwise, the contents of the output table columns may vary from run to run.

**PivotColumn**

[Required if you omit NumberOfRows.] Specify the name of the input column that contains the pivot keys.

If *pivot\_column* contains numeric values, the function casts them to VARCHAR; function performance improves slightly if you specify NumericPivotKey ('true').

**PivotKeys**

[Required if you specify PivotColumn.] Specify the values in *pivot\_column* to use as pivot keys. The function ignores rows that contain other values in *pivot\_column*.

**NumericPivotKey**

[Optional] Use this syntax element only with the PivotColumn syntax element. If *pivot\_column* is numeric, NumericPivotKey ('true') improves function performance slightly.

Default: 'false'

**TargetColumns**

[Optional] Specify the names of the target columns (input columns that contain the values to pivot).

## Pivoting Input

**Input Table Schema**

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which to partition input data.
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Values to pivot.

## Pivoting Output

The output table schema depends on whether you specify the syntax element NumberOfRows or PivotColumn.

**Output Table Schema, NumberOfRows**

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned.
<i>value_i</i>	Any	[Column appears <i>number_of_rows</i> times.] Value in <i>i</i> th target column, where <i>i</i> is in range [0, <i>number_of_rows</i> -1]. Columns appear in order specified by ORDER BY clause.

**Output Table Schema, PivotColumn**

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned.
<i>value_target_column</i>	Any	[Column appears once for each <i>pivot_key</i> .] Values for <i>pivot_key</i> that are associated with partitions in row.

## Pivoting Examples

### Pivoting Example: NumberOfRows

This example specifies the NumberOfRows syntax element.

#### Input

The input table, `pivot_input`, contains temperature, pressure, and dewpoint data for three cities, in sparse format.

**pivot\_input**

sn	city	week	attribute	value1
1	Asheville	1	temp	32
1	Asheville	1	pressure	1020.8
1	Asheville	1	dewpoint	27.6F
2	Asheville	2	temp	32
2	Asheville	2	pressure	1021.3
2	Asheville	2	dewpoint	27.4F
3	Asheville	3	temp	34
3	Asheville	3	pressure	1021.7
3	Asheville	3	dewpoint	28.2F
4	Nashville	1	temp	42
4	Nashville	1	pressure	1021
4	Nashville	1	dewpoint	29.4F
5	Nashville	2	temp	44
5	Nashville	2	pressure	1019.8

sn	city	week	attribute	value1
5	Nashville	2	dewpoint	29.2F
6	Brownsville	2	temp	47
6	Brownsville	2	pressure	1019
6	Brownsville	2	dewpoint	28.9F
7	Brownsville	3	temp	46
7	Brownsville	3	pressure	1019.2
7	Brownsville	3	dewpoint	28.9F

## SQL Call

```
SELECT * FROM Pivoting (
  ON pivot_input PARTITION BY sn,city,week
  ORDER BY week,attribute
  USING
  PartitionColumns ('sn','city', 'week')
  NumberOfRows (3)
  TargetColumns ('value1')
) AS dt ORDER BY 1,2,3;
```

The ORDER BY clause is required. If omitted, the output table column content is nondeterministic (for more information, see [Nondeterministic Results and UniqueID Syntax Element](#)). The function adds any NULL values at the end.

## Output

The function outputs the input column contents in dense format in the output columns value1\_0, value1\_1, and value1\_2, which contain the dewpoint, pressure, and temperature, respectively. Because these values are numeric, the function casts them to VARCHAR.

sn	city	week	value1_0	value1_1	value1_2
1	asheville	1	27.6f	1020.8	32
2	asheville	2	27.4f	1021.3	32
3	asheville	3	28.2f	1021.7	34
4	nashville	1	29.4f	1021	42
5	nashville	2	29.2f	1019.8	44
6	brownsville	2	28.9f	1019	47
7	brownsville	3	28.9f	1019.2	46

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Pivoting Example: PivotKeys

### Input

- Input table: pivot\_input, as in [Pivoting Example: NumberOfRows](#)

### SQL Call

```
SELECT * FROM Pivoting (
  ON pivot_input PARTITION BY sn,city,week
  USING
  PartitionColumns ('sn','city', 'week')
  PivotKeys ('temp','pressure')
  PivotColumn ('attribute')
  TargetColumns ('value1')
) AS dt ORDER BY 1,2,3;
```

With PivotKeys, the function does not use the ORDER BY clause.

### Output

To create the output table, the function pivots the input table on the partition columns (sn, city, and week) and outputs the contents of the target column (value1) in dense format in the output columns value1\_pressure and value1\_temp.

sn	city	week	value1_pressure	value1_temp
1	asheville	1	1020.8	32
2	asheville	2	1021.3	32
3	asheville	3	1021.7	34
4	nashville	1	1021	42
5	nashville	2	1019.8	44
6	brownsville	2	1019	47
7	brownsville	3	1019.2	46

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Description - Normalization (Scale and ScaleMap)

- In statistics and applications of statistics, Normalization can have a range of meanings. In the simplest cases, normalization of ratings means **adjusting values measured on different scales to a notionally common scale**, often prior to averaging
- In more complicated cases, Normalization may refer to more sophisticated adjustments where the intention is to bring the entire probability distributions of adjusted values into alignment. In the case of normalization of scores in educational assessment, there may be an intention to align distributions to a normal distribution

High-bias ML algorithms (like Linear Regression, Logistic Regression, Kmeans) can underfit Model; i.e., can't make accurate Predictions on either Train or Test set.  
Normalization can minimize this tendency

Normalization uses **Feature scaling** which is a method used to standardize the range of independent variables or features of data.

In [data processing](#), it is also known as data normalization and is generally performed during the data preprocessing step.

## Why Used - Normalization

- The black box answer is you can't train models when your features have different ranges (1-5 vs 1-5000)
- In essence, Normalization is done to have the same range of values for each of the inputs to the Model. This can guarantee stable convergence of weight and **biases**

id	room area	room number	height	price
1	100	3	2.6	200,000
2	150	4	3	300,000
...	...	...	...	...

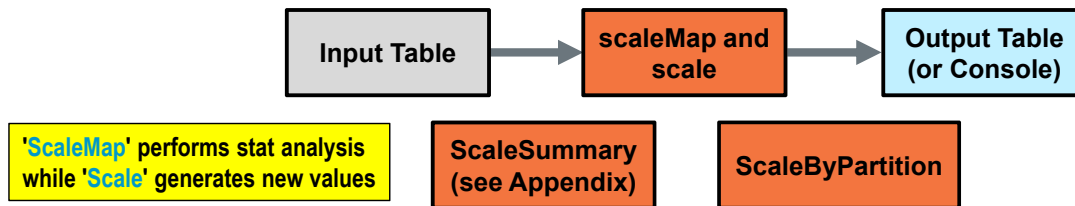
If one of the features has a broad range of values, the distance will be governed by this particular feature

Range in column 'room area' is 50 and it is significantly larger than the range in column 'height'. So we can't compare them directly

Since the range of values of raw data varies widely, in some [machine learning](#) algorithms, objective functions will not work properly without [normalization](#).

For example, the majority of [classifiers](#) calculate the distance between two points by the [Euclidean distance](#). If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance. Another reason why feature scaling is applied is that [gradient descent](#) converges much faster with feature scaling than without it.<sup>1</sup>

## Workflow - Normalization



Function	Description
<b>ScaleMap</b>	Takes data set and outputs its statistical information (assembled at vworker level)
<b>Scale</b>	Takes ScaleMap output and outputs scaled (normalized) values for input data set. You can use Scale output as input to distance-based analysis functions, such as KMeans
<b>ScaleSummary</b>	Takes ScaleMap output and outputs global statistical information for the entire input data set
<b>ScaleByPartition</b>	Scales sequences in each partition, using same formula as Scale

Here are the four functions associated with Normalization:

- ScaleMap and Scale
- ScaleSummary
- ScaleByPartition

## Syntax - Scale and ScaleMap

```

SELECT * FROM Scale
(ON { table | view | (query) } AS "input" PARTITION BY ANY
ON (
SELECT * FROM ScaleMap (
ON { table | view | (query) }
USING
TargetColumns ( { 'target_column' | target_column_range }[,...] )
[ MissValue ( { 'KEEP' | 'OMIT' | 'ZERO' | 'LOCATION' } ) ]
) AS alias_1
) AS statistic DIMENSION
USING
ScaleMethod ( 'method' [,...] )
[ GlobalScale ( { 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' } ) ]
[ TargetColumns ( { 'target_column' | target_column_range }[,...] ) ]
[ Accumulate ( { 'accumulate_column' | accumulate_column_range }[,...] ) ]
[ Multiplier (multiplier [,...] ) ]
[ Intercept (intercept [,...] ) ]
) AS dt;

```

**ScaleMap** performs stat analysis while **Scale** generates new values

**'Method'** tells you how to scale variables

Here's the generic syntax for the Outlier function. The next few slides will cover the individual arguments in detail.

## Arguments - ScaleMap

- **TargetColumns** Specify the names of input table columns for which to calculate statistics. The columns must contain numeric values
- **MissValue** [Optional] Specify how the Scale, ScaleMap, and ScaleByPartition functions are to process NULL values in input, as follows:

Option	Description
<b>Keep(Default)</b>	Keep NULL values
<b>Omit</b>	Ignore any row that has a NULL value
<b>Zero</b>	Replace each NULL value with zero
<b>Location</b>	Replace each NULL value with its location value. <b>Note:</b> Location definition varies by Method; e.g., for Method "midrange", defined as $(\max X + \min X) / 2$

```
USING  
TargetColumns ('[2:6]')  
MissValue ('omit')
```

The 'MissValue' argument has four options which determine how NULL values are handled.

## Arguments - Scale (1 of 3)

- **Method** Specify one or more statistical methods to Scale the data set

Method	Location	Scale
<b>mean</b>	Xmean	1
<b>sum</b>	0	$\Sigma X$
<b>ustd</b>	0	Standard deviation, calculated according to biased estimator of variance
<b>std</b>	Xmean	Standard deviation, calculated according to unbiased estimator of variance
<b>range</b>	minx	maxX - minX
<b>midrange</b>	(maxx+minx)/2	(maxX - minX)/2

There are a number of statistical algorithms that can be used with the 'Scale' function as shown in the slide.

## Arguments - Scale (2 of 3)

- **Global** [Optional] Specify whether all input columns scaled to same location and scale
- **TargetColumns** [Optional] Specify columns that contain values to scale.  
Default: All columns from ScaleMap except sttype
- **Accumulate** [Optional] Specify the input table columns to copy to the output table
- **Multiplier** [Optional] Specify one or more multiplying factors to apply to the input variables
  - multiplier in the following formula:

$$X' = \text{intercept} + \text{multiplier} * (X - \text{location})/\text{scale}$$

Default: multiplier is 1

## Arguments - Scale (3 of 3)

**Intercept** [Optional] Specify one or more addition factors incrementing the scaled results - intercept in the following formula:

$$X' = \text{intercept} + \text{multiplier} * (X - \text{location})/\text{scale}$$

If you specify only one intercept, it applies to all columns specified by the TargetColumns argument. If you specify multiple addition factors, each intercept applies to the corresponding input column. This is the syntax of intercept:

`[-]{number | min | mean | max }`

where min, mean, and max are the scaled global minimum, maximum, mean values of the corresponding columns. This is the formula for computing the scaled global minimum:  $\text{scaledmin} = (\text{minX} - \text{location})/\text{scale}$  Default: intercept is 0

## Data We'll Be Using

The Input variables are as follows

```
SELECT * FROM scale_housing;
```

0	1	2	3	4	5	6
types	id	price	lotsize	bedrooms	bathrms	stories
classic	1	42000.0	5850.0	3.0	1.0	2.0
classic	2	null	4000.0	2.0	1.0	1.0
classic	3	49500.0	3060.0	3.0	1.0	1.0
classic	4	60500.0	6650.0	3.0	1.0	2.0
classic	5	61000.0	6360.0	2.0	1.0	1.0
bungalow	6	66000.0	4160.0	3.0	1.0	1.0
bungalow	7	66000.0	3880.0	null	2.0	2.0
bungalow	8	60000.0	4160.0	3.0	1.0	2.0

Here's the data we'll be using for the next lab.



teradata.

## Lab 05: Scale and ScaleMap (Method='midrange')

Output

id	price	lotsize	bedrooms	bathrms	stories
1	-1.0	0.5543175487465181	1.0	-1.0	-0.3333333333333333
3	-0.6774193548387096	-1.0	1.0	-1.0	-1.0
4	-0.20430107526881722	1.0	1.0	-1.0	-0.3333333333333333
5	-0.1827956989247312	0.8384401114206128	-1.0	-1.0	-1.0
6	0.03225806451612903	-0.3871866295264624	1.0	-1.0	-1.0
8	0.16129032258064516	-0.3871866295264624	1.0	-1.0	0.3333333333333333
9	0.7978494623655914	-0.03064066852367688	1.0	-1.0	-1.0
10	1.0	0.3593314763231198	1.0	1.0	1.0

```

2 SELECT * FROM Scale
  (ON ScaleMap
  (ON scale_housing
1  USING
   TargetColumns ('[2:6]')
   MissValue ('omit')
   ) AS statistic DIMENSION
   ON scale_housing AS "input" PARTITION BY ANY
2  USING
   "Method" ('midrange')
   Accumulate ('id')
   ) AS dt
ORDER BY id, price, lotsize;

```

Ignore any row with NULL

Output from ScaleMap (Input for Scale)

How to Scale variables

Put Column in Output

-- (maxX - minX)/2

'ScaleMap' performs stat analysis  
while 'Scale' generates new values

Here we normalize the data using Method = 'midrange'.



## Lab 06: Scale and ScaleMap (Multiple Method)

teradata.

Output

```
SELECT * FROM Scale
(ON ScaleMap
(ON scale_housing
USING
TargetColumns ('[2:6]')
MissValue ('omit')
) AS statistic DIMENSION
ON scale_housing AS "input" PARTITION BY ANY
USING
"Method" ('midrange', 'mean', 'maxabs', 'range')
Accumulate ('id')
) AS dt
ORDER BY id, price, lotsize;
```

id	price	lotsize	bedrooms	bathrms	stories	scalemethod
1	-23037.5	782.5	0.125	-0.125	0.125	mean
1	-1.0	0.554317548...	1.0	-1.0	-0.3333...	midrange
1	0.0	0.777158774...	1.0	0.0	0.33333...	range
1	0.47457627118...	0.879699248...	1.0	0.5	0.5	maxabs
3	-15537.5	-2007.5	0.125	-0.125	-0.875	mean
3	0.6774193548	1.0	1.0	1.0	1.0	midrange

Ignore any row with NULL

Output from ScaleMap (Input for Scale)

Here we normalize the data using four different 'Methods'. The Output displays all four.



## Lab 07: Scale and ScaleMap (Method='maxabs')

teradata.

```
CREATE MULTISET TABLE pc_normalized AS
(SELECT * FROM Scale
(ON computers_train1 AS INPUT PARTITION BY ANY
ON(SELECT * FROM ScaleMap
(ON computers_train1
USING
TargetColumns ('[1:5]')
MissValue ('omit') ← Ignore any row with NULL
) AS dt1
) AS statistic DIMENSION
USING
"Method" ('maxabs') ← All Output values between 0
Accumulate ('id')
) AS dt2
ORDER BY id
) WITH DATA;
```

### Input

	1	2	3	4	5
id	price	speed	hd	ram	screen
2667	2590.0	66.0	426.0	8.0	14.0
818	1848.0	33.0	130.0	4.0	14.0
2158	1495.0	33.0	212.0	4.0	14.0
154	2075.0	33.0	250.0	8.0	14.0
3747	1999.0	33.0	527.0	4.0	14.0

### Output

id	price	speed	hd	ram	screen
4387	0.4628634932...	0.66	0.25095...	0.5	0.882352...
458	0.4426745693...	0.66	0.10190...	0.125	0.882352...
4453	0.4158177440...	0.33	0.34285...	0.5	0.823529...
1877	0.3415447305...	0.25	0.10190...	0.125	0.823529...
302	0.3500000000...	0.35	0.05100...	0.125	0.823529...

And, finally we Normalize using Method = 'maxabs'.

## Description and Syntax - ScaleByPartition

The **ScaleByPartition** function scales the sequences in each partition independently, using the same formula as the function **Scale**

```
SELECT * FROM ScaleByPartition
(ON { table | view | (query) } PARTITION BY partition_columns
USING
  "Method" ('method' [,...])
  [ MissValue ( { 'KEEP' | 'OMIT' | 'ZERO' | 'LOCATION' } ) ]
  TargetColumns ( { 'input_column' | input_column_range } [,...] )
  [ "Global" ( { 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' } ) ]
  [ Accumulate ( { 'accumulate_column' | accumulate_column_range } [,...] ) ]
  [ Multiplier ('multiplier' [,...]) ]
  [ Intercept ('intercept' [,...]) ]
) AS alias;
```

If your data has Partitions (like we did earlier with **OutlierFilter** (Temperature and Pressure), you can 'group' these partitions via **ScaleByPartition** function

If your Input data has partitions, you can use the ScaleByPartition function instead of the Scale function.

## Data We'll Be Using

```
SELECT * FROM scale_housing;
```

We will Partition by the 'types' column in our housing dataset

	0	1	2	3	4	5	6
	types	id	price	lotsize	bedrooms	bathrms	stories
	classic	1	42000.0	5850.0	3.0	1.0	2.0
	classic	2	null	4000.0	2.0	1.0	1.0
	classic	3	49500.0	3060.0	3.0	1.0	1.0
	classic	4	60500.0	6650.0	3.0	1.0	2.0
	classic	5	61000.0	6360.0	2.0	1.0	1.0
	bungalow	6	66000.0	4160.0	3.0	1.0	1.0
	bungalow	7	66000.0	3880.0	null	2.0	2.0
	bungalow	8	60000.0	4160.0	2.0	1.0	2.0

Here's the data we'll be using for the next lab.



## Lab 08: ScaleByPartition

```
SELECT * FROM ScaleByPartition
(ON scale_housing PARTITION BY types
USING
TargetColumns ('[2:6]')
"Method" ('maxabs')
Accumulate ('types', 'id')
) AS dt
ORDER BY 1 desc,2;
```

Output

types	id	price	lotsize	bedrooms	bathrms	stories
classic	1	0.68852459...	0.8796992...	1.0	1.0	1.0
classic	2	null	0.6015037...	0.6666666...	1.0	0.5
classic	3	0.81147540...	0.4601503...	1.0	1.0	0.5
classic	4	0.99180327...	1.0	1.0	1.0	1.0
classic	5	1.0	0.9563909...	0.6666666...	1.0	0.5
bungalow	6	0.74576271...	0.7563636...	1.0	0.5	0.25
bungalow	7	0.74576271...	0.7054545...	null	1.0	0.5
bungalow	8	0.77966101...	0.7563636...	1.0	0.5	0.75
bungalow	9	0.94689265...	0.8727272...	1.0	0.5	0.25
bungalow	10	1.0	1.0	1.0	1.0	1.0

Recall without Partitions, we used the 'PARTITION BY ANY' argument

```
SELECT * FROM Scale (
ON ScaleMap (
ON scale_housing
USING
TargetColumns ('[2:6]')
MissValue ('omit')
) AS statistic DIMENSION
ON scale_housing AS "input" PARTITION BY ANY
USING
"Method" ('midrange')
Accumulate ('id')
) AS dt ORDER BY id, price, lotsize;
```

The Output show the statistics from the Input based on the two partitions, Classic and Bungalow.

# CASE Expressions

## Overview

The following sections describe SQL CASE expressions.

## CASE

### Purpose

Specifies alternate values for a conditional expression or expressions based on equality comparisons and conditions that evaluate to TRUE.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Overview

CASE provides an efficient and powerful method for application developers to change the representation of data, permitting conversion without requiring host program intervention.

For example, you could code employee status as 1 or 2, meaning full-time or part-time, respectively. For efficiency, the system stores the numeric code but prints or displays the appropriate textual description in reports. This storage and conversion is managed by Teradata Database.

In addition, CASE permits applications to generate nulls based on information derived from the database, again without host program intervention. Conversely, CASE can be used to convert a null into a value.

## Two Forms of CASE Expressions

CASE expressions are specified in these forms:

- Valued CASE is described under “Valued CASE Expression”.
- Searched CASE is described under “Searched CASE Expression”.

## CASE Shorthands for Handling Nulls

Two shorthand forms of CASE are provided to handle nulls:

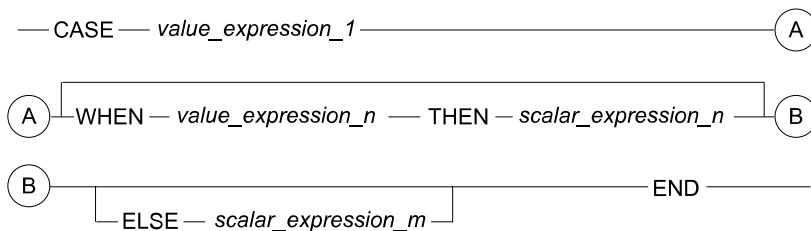
- COALESCE is described under “COALESCE Expression”.
- NULLIF is described under “NULLIF Expression”.

## Valued CASE Expression

### Purpose

Evaluates a set of expressions for equality with a test expression and returns as its result the value of the scalar expression defined for the first WHEN clause whose value equals that of the test expression. If no equality is found, then CASE returns the scalar value defined by an optional ELSE clause, or if omitted, NULL.

### Syntax



### Syntax Elements

#### *value\_expression\_1*

An expression whose value is tested for equality with *value\_expression\_n*.

#### *value\_expression\_n*

A set of expressions against which the value for *value\_expression\_1* is tested for equality.

#### *scalar\_expression\_n*

An expression whose value is returned on the first equality comparison of *value\_expression\_1* and *value\_expression\_n*.

#### *scalar\_expression\_m*

An expression whose value is returned if evaluation falls through to the ELSE clause.

### ANSI Compliance

This statement is ANSI SQL:2011 compliant.

Teradata Database does not enforce the ANSI restriction that *value\_expression\_1* must be a deterministic function. In particular, Teradata Database allows the function RANDOM to be used in *value\_expression\_1*.

Note that if RANDOM is used, nondeterministic behavior may occur, depending on whether *value\_expression\_1* is recalculated for each comparison to *value\_expression\_n*.

## Usage Notes

WHEN clauses are processed sequentially.

The first WHEN clause *value\_expression\_n* that equates to *value\_expression\_1* returns the value of its associated *scalar\_expression\_n* as its result. The evaluation process then terminates.

If no *value\_expression\_n* equals *value\_expression\_1*, then *scalar\_expression\_m*, the argument of the ELSE clause, is the result.

If no ELSE clause is defined, then the result defaults to NULL.

The data type of *value\_expression\_1* must be comparable with the data types of all of the *value\_expression\_n* values.

For information on the result data type of a CASE expression, see [Rules for the CASE Expression Result Type](#).

You can use a scalar subquery in the WHEN clause, THEN clause, and ELSE clause of a CASE expression. If you use a non-scalar subquery (a subquery that returns more than one row), a runtime error is returned.

Recommendation: Do not use the built-in functions CURRENT\_DATE or CURRENT\_TIMESTAMP in a CASE expression that is specified in a partitioning expression for a partitioned primary index (PPI). In this case, all rows are scanned during reconciliation.

## Default Title

The default title for a CASE expression appears as:

```
<CASE expression>
```

## Restrictions on the Data Types in a CASE Expression

The following restrictions apply to CLOB, BLOB, and UDT types in a CASE expression:

Data Type	Restrictions
BLOB	A BLOB can only appear in <i>value_expression_1</i> , <i>value_expression_n</i> , <i>scalar_expression_m</i> , or <i>scalar_expression_n</i> when it is cast to BYTE or VARBYTE.
CLOB	A CLOB can only appear in <i>value_expression_1</i> , <i>value_expression_n</i> , <i>scalar_expression_m</i> , or <i>scalar_expression_n</i> when it is cast to CHAR or VARCHAR.
UDT	Multiple UDTs can appear in a CASE expression, with the following restrictions:

Data Type	Restrictions
	<ul style="list-style-type: none"> <li>The data type of value_expression_1 through value_expression_n must have the same UDT data type if one of them has a UDT data type.</li> <li>scalar_expression_n and scalar_expression_m must be the same UDT data type if one of them has a UDT data type.</li> </ul> <p>Teradata Database does not perform implicit type conversion on UDTs in CASE expressions. A workaround for this restriction is to use CREATE CAST to define casts that cast between the UDTs, and then explicitly invoke the CAST function in the CASE expression. For more information on CREATE CAST, see <i>Teradata Vantage™ SQL Data Definition Language Syntax and Examples</i>, B035-1144.</p>

## Examples

### Example: Calculating the Fraction of Cost

The following example uses a Valued CASE expression to calculate the fraction of cost in the total cost of inventory represented by parts of type '1':

```
SELECT SUM(CASE part
            WHEN '1'
            THEN cost
            ELSE 0
            END
        )/SUM(cost)
FROM t;
```

### Example: Using a CASE Expression

A CASE expression can be used in place of any *value-expression*.

```
SELECT *
FROM t
WHERE x = CASE y
            WHEN 2
            THEN 1001
            WHEN 5
            THEN 1002
            END;
```

## Example: Combining a CASE Expression with a Concatenation Operator

The following example shows how to combine a CASE expression with a concatenation operator:

```
SELECT prodID, CASE prodSTATUS
                WHEN 1
                THEN 'SENT'
                ELSE 'BACK ORDER'
                END || ' STATUS'
FROM t1;
```

## Example: Using UDT Data Types in Value Expressions

You use *value\_expression\_1* through *value\_expression\_n* to test for equality in a valued CASE expression.

For these examples, the table is defined as follows:

```
create table udtval038_t1(id integer, udt1 testcircleudt, udt2 testrectangleudt)
PRIMARY INDEX (id);
```

The following example shows a valued CASE expression, where all value expressions are of the same UDT data type:

```
SELECT CASE udt1
        WHEN new testcircleudt('1,1,2,yellow,circ')
        THEN 'Row 1'
        WHEN new testcircleudt('2,2,4,purple,circ')
        THEN 'Row 2'
        WHEN new testcircleudt('3,3,9,green,circ')
        THEN 'Row 3'
        ELSE 'Row is NULL'
        END
FROM t1;
*** Query completed. 4 rows found. One column returned.
<CASE expression>
-----
Row 3
Row 1
Row is NULL
Row 2
```

However, the following example does not complete successfully because *testrectangleudt* does not match the other UDT data types:

```

SELECT CASE udt1
    WHEN new testcircledt('1,1,2,yellow,circ')
    THEN 'Row 1'
    WHEN new testrectangleudt('2,2,4,4,purple,rect')
    THEN 'Row 2'
    WHEN new testcircledt('3,3,9,green,circ')
    THEN 'Row 3'
    ELSE 'Row is NULL'
    END
FROM t1;

```

### Example 1: Using UDT Data Types in Scalar Expressions

You use *scalar\_expression\_n* and *scalar\_expression\_m* as the expressions to return on when the equality comparison on a valued or searched CASE expression evaluates to TRUE, or the value to return on in an ELSE condition.

For these examples, the table is defined as follows:

```

create table udtval038_t1(id integer, udt1 testcircledt, udt2 testrectangleudt)
PRIMARY INDEX (id);

```

Following is an example of a searched CASE Expression where all scalar expressions are of the same UDT data type.

#### Note:

The *search\_condition\_n* can be a different UDT data type than the *scalar\_expression\_n*.  
 SELECT  
 \* FROM udtval038\_t1

```

WHERE udt1 = CASE
    WHEN udt2 <> new testrectangleudt('2,2,4,4,pink,rect')
    THEN new testcircledt('1,1,2,blue,circ')
    ELSE new testcircledt('2,2,4,purple,circ')
*** Query completed. 2 rows found. 3 columns returned.
END;
id udt1
-----
1 1, 1, 2, yellow, circ
2 2, 2, 4, purple, circ

```

However, the following example does not complete successfully because the scalar expressions are of different data types.

```

SELECT * FROM udtval038_t1
WHERE udt1 = CASE

```

```

WHEN udt2 <> new testrectangleudt('2,2,4,4,pink,rect')
THEN new testcircleudt('1,1,2,blue,circ')
ELSE new testrectangleudt('2,2,4,4,purple,rect')
END;

```

## Related Topics

For more information, see:

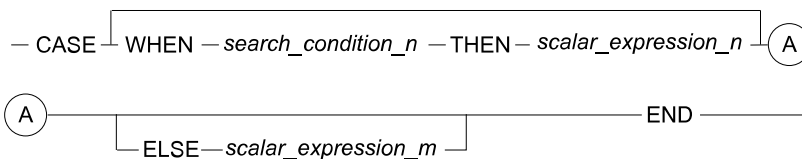
- For information about error conditions, see [Error Conditions](#).
- For information about the result data type of a CASE expression, see [Rules for the CASE Expression Result Type](#).
- For information about format of the result of a CASE expression, see [Default Format](#).
- For information about nulls and CASE expressions, see [CASE and Nulls](#).

## Searched CASE Expression

### Purpose

Evaluates a search condition and returns one of a WHEN clause-defined set of scalar values when it finds a value that evaluates to TRUE. If no TRUE test is found, then CASE returns the scalar value defined by an ELSE clause, or if omitted, NULL.

### Syntax



## Syntax Elements

### *search\_condition\_n*

A predicate condition to be tested for truth.

### *scalar\_expression\_n*

A scalar expression whose value is returned when *search\_condition\_n* is the first search condition that evaluates to TRUE.

*scalar\_expression\_m*

A scalar expression whose value is returned when no *search\_condition\_n* evaluates to TRUE.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Usage Notes

WHEN clauses are processed sequentially.

The first WHEN clause *search\_condition\_n* that is TRUE returns the value of its associated *scalar\_expression\_n* as its result. The evaluation process then ends.

If no *search\_condition\_n* is TRUE, then *scalar\_expression\_m*, the argument of the ELSE clause, is the result.

If no ELSE clause is defined, then the default value for the result is NULL.

You can use a scalar subquery in the WHEN clause, THEN clause, and ELSE clause of a CASE expression. If you use a non-scalar subquery (a subquery that returns more than one row), a runtime error is returned.

Recommendation: Do not use the built-in functions CURRENT\_DATE or CURRENT\_TIMESTAMP in a CASE expression that is specified in a partitioning expression for a partitioned primary index (PPI). In this case, all rows are scanned during reconciliation.

## Default Title

The default title for a CASE expression appears as:

```
<CASE expression>
```

## Rules for WHEN Search Conditions

WHEN search conditions have the following properties:

- Can take the form of any comparison operator, such as LIKE, =, or <>.
- Can be a quantified predicate, such as ALL or ANY.
- Can contain a scalar subquery.
- Can contain joins of two tables.

For example:

```
SELECT CASE
  WHEN t1.x=t2.x THEN t1.y
```

```
ELSE t2.y
END FROM t1,t2;
```

- Cannot contain SELECT statements.

## Restrictions on the Data Types in a CASE Expression

The following restrictions apply to CLOB, BLOB, and UDT types in a CASE expression:

Data Type	Restrictions
BLOB	A BLOB can only appear in <i>value_expression_1</i> , <i>value_expression_n</i> , <i>scalar_expression_m</i> , or <i>scalar_expression_n</i> when it is cast to BYTE or VARBYTE.
CLOB	A CLOB can only appear in <i>value_expression_1</i> , <i>value_expression_n</i> , <i>scalar_expression_m</i> , or <i>scalar_expression_n</i> when it is cast to CHAR or VARCHAR.
UDT	<p>Multiple UDTs can appear in a CASE expression, with the following restrictions:</p> <ul style="list-style-type: none"> <li>• The data type of <i>value_expression_1</i> through <i>value_expression_n</i> must have the same UDT data type if one of them has a UDT data type.</li> <li>• <i>scalar_expression_n</i> and <i>scalar_expression_m</i> must be the same UDT data type if one of them has a UDT data type.</li> </ul> <p>Teradata Database does not perform implicit type conversion on UDTs in CASE expressions. A workaround for this restriction is to use CREATE CAST to define casts that cast between the UDTs, and then explicitly invoke the CAST function in the CASE expression. For more information on CREATE CAST, see <i>Teradata Vantage™ SQL Data Definition Language Syntax and Examples</i>, B035-1144.</p>

## Examples

### Example: Evaluating a Search Condition

The following statement is equivalent to the first example of the valued form of CASE on “Example”:

```
SELECT SUM(CASE
            WHEN part='1'
            THEN cost
            ELSE 0
            END
          ) / SUM(cost)
FROM t;
```

## Example: Using a CASE Expression

CASE expressions can be used in place of any *value-expressions*.

Note that the following example does not specify an ELSE clause. ELSE clauses are always optional in a CASE expression. If an ELSE clause is not specified and none of the WHEN conditions are TRUE, then a null is returned.

```
SELECT *
FROM t
WHERE x = CASE
            WHEN y=2
            THEN 1
            WHEN (z=3 AND y=5)
            THEN 2
            END;
```

## Example: Using an ELSE Clause

The following example uses an ELSE clause.

```
SELECT *
FROM t
WHERE x = CASE
            WHEN y=2
            THEN 1
            ELSE 2
            END;
```

## Example: Using a CASE expression to Enhance Performance

The following example shows how using a CASE expression can result in significantly enhanced performance by eliminating multiple passes over the data. Without using CASE, you would have to perform multiple queries for each region and then consolidate the answers to the individual queries in a final report.

```
SELECT SalesMonth, SUM(CASE
                      WHEN Region='NE'
                      THEN Revenue
                      ELSE 0
                      END),
               SUM(CASE
                      WHEN Region='NW'
```

```

        THEN Revenue
        ELSE 0
      END),
    SUM(CASE
      WHEN Region LIKE 'N%'
      THEN Revenue
      ELSE 0
    END)
  AS NorthernExposure, NorthernExposure/SUM(Revenue),
  SUM(Revenue)
FROM Sales
GROUP BY SalesMonth;

```

## Example: Producing a Report to Show Employee Salary

All employees whose salary is less than \$40000 are eligible for an across the board pay increase.

IF your salary is less than ...	AND you have greater than this many years of service ...	THEN you receive this percentage salary increase ...
\$30000.00	8	15
\$35000.00	10	10
\$40000.00		5

The following SELECT statement uses a CASE expression to produce a report showing all employees making under \$40000, displaying the first 15 characters of the last name, the salary amount (formatted with \$ and punctuation), the number of years of service based on the current date (in the column named On\_The\_Job) and which of the four categories they qualify for: '15% Increase', '10% Increase', '05% Increase' or 'Not Qualified'.

```

SELECT CAST(last_name AS CHARACTER(15))
, salary_amount (FORMAT '$,$9,999.99')
, (date - hire_date)/365.25 (FORMAT 'Z9.99') AS On_The_Job
, CASE
  WHEN salary_amount < 30000 AND On_The_Job > 8
  THEN '15% Increase'
  WHEN salary_amount < 35000 AND On_The_Job > 10
  THEN '10% Increase'
  WHEN salary_amount < 40000 AND On_The_Job > 10
  THEN '05% Increase'
  ELSE 'Not Qualified'
END AS Plan
WHERE salary_amount < 40000

```

```
FROM employee
ORDER BY 4;
```

The result of this query appears in the following table:

last_name	salary_amount	On_The_Job	Plan
Trader	\$37,850.00	20.61	05% Increase
Charles	\$39,500.00	18.44	05% Increase
Johnson	\$36,300.00	20.41	05% Increase
Hopkins	\$37,900.00	19.99	05% Increase
Morrissey	\$38,750.00	18.44	05% Increase
Ryan	\$31,200.00	20.41	10% Increase
Machado	\$32,300.00	18.03	10% Increase
Short	\$34,700.00	17.86	10% Increase
Lombardo	\$31,000.00	20.11	10% Increase
Phillips	\$24,500.00	19.95	15% Increase
Rabbit	\$26,500.00	18.03	15% Increase
Kanieski	\$29,250.00	20.11	15% Increase
Hoover	\$25,525.00	20.73	15% Increase
Crane	\$24,500.00	19.15	15% Increase
Stein	\$29,450.00	20.41	15% Increase

## Related Topics

For more information, see:

- For information about error conditions, see [Error Conditions](#).
- For information about the result data type of a CASE expression, see [Rules for the CASE Expression Result Type](#).
- For information about format of the result of a CASE expression, see [Default Format](#).
- For information about nulls and CASE expressions, see [CASE and Nulls](#).

## Error Conditions

The following conditions or expressions are considered illegal in a CASE expression:

Condition or Expression	Example
A condition after the keyword CASE is supplied.	<pre> SELECT CASE a=1       WHEN 1       THEN 1       ELSE 0       END FROM t; </pre>
A non valid WHEN expression is supplied in a valued CASE expression.	<pre> SELECT CASE a       WHEN a=1       THEN 1       ELSE 0       END FROM t; </pre>
A non valid WHEN condition is supplied in a searched CASE expression.	<pre> SELECT CASE       WHEN a       THEN 1       ELSE 0       END FROM t; SELECT CASE       WHEN NULL       THEN 'NULL'       END FROM table_1; </pre>
A non-scalar subquery is specified in a WHEN condition of a searched CASE expression.	<pre> SELECT CASE       WHEN t.a IN         (SELECT u.a          FROM u)       THEN 1       ELSE 0       END FROM t; </pre>
A CASE expression references multiple UDTs that are not identical to each other.	<pre> SELECT CASE t.shape.gettype()       WHEN 1       THEN NEW circle('18,18,324')       WHEN 2       THEN NEW square('20,20,400')       END; </pre>

## Rules for the CASE Expression Result Type

Because the expressions in CASE THEN/ELSE clauses can be different data types, determining the result type is not always straightforward. You can use the TYPE attribute function with the CASE expression as the argument to find out the result data type. See [TYPE](#).

The following rules apply to the data type of the CASE expression result.

## THEN/ELSE Expressions Having the Same Non-Character Data Type

If all of the THEN and ELSE expressions have the same non-character data type, the result of the CASE expression is that type. For example, if all of the THEN and ELSE expressions have an INTEGER type, the result type of the CASE expression is INTEGER.

For information about how the precision and scale of DECIMAL results are calculated, see [Binary Arithmetic Result Data Types](#).

## THEN/ELSE Character Type Expressions

The following rules apply to CASE expressions where the data types of all of the THEN/ELSE expressions are character:

- The result of the CASE expression is also a character data type, with the length equal to the maximum length of the different character data types of the THEN/ELSE expressions.
- If the data types of all of the THEN/ELSE expressions are CHARACTER (or CHAR), the result data type is CHARACTER. If one or more expressions are VARCHAR (or LONG VARCHAR), the result data type is VARCHAR.
- The server character set of the result is determined as follows:
  - If the CASE expression contains 1 nonliteral character expression and 1 or more literals, then Teradata Database tries to translate every literal to the character set of the nonliteral. If the translations are successful, then the character set of the nonliteral is used for the result data type. If the translations are not successful, the server character set of the result is Unicode.
  - If the CASE expression contains more than 1 nonliteral character expression and 1 or more literals, then:

If all of the nonliteral expressions have the same character set, then Teradata Database uses this character set as the common data type. Otherwise, if the nonliteral expressions have differing character sets, then Teradata Database uses the Unicode character set as the common data type.

Teradata Database tries to translate every literal to the character set of the common data type. If the translations are successful, then the result data type has the character set of the common data type. If the translations are not successful, the server character set of the result is Unicode.

## Examples

### Examples of Character Data in a CASE Expression

For the following examples of CHARACTER data behavior, assume the default server character set is KANJI1 and the table definition for the CASE examples is as follow:

```
CREATE TABLE table_1
(
  i          INTEGER,
  column_l  CHARACTER(10) CHARACTER SET LATIN,
  column_u  CHARACTER(10) CHARACTER SET UNICODE,
  column_j  CHARACTER(10) CHARACTER SET KANJISJIS,
  column_g  CHARACTER(10) CHARACTER SET GRAPHIC,
  column_k  CHARACTER(10) CHARACTER SET KANJI1
);
```

**Note:**

In accordance with Teradata internationalization plans, KANJI1 support is deprecated and is to be discontinued in the near future. KANJI1 is not allowed as a default character set; the system changes the KANJI1 default character set to the UNICODE character set. Creation of new KANJI1 objects is highly restricted. Although many KANJI1 queries and applications may continue to operate, sites using KANJI1 should convert to another character set as soon as possible. For more information, see "KANJI1 Character Set" in *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

## Examples of Character Data in a CASE Expression: Example 1

The server character set of the result of the following query is UNICODE because the CASE expression contains more than 1 nonliteral character expressions with differing character sets.

```
SELECT i, CASE
      WHEN i=2 THEN column_u
      WHEN i=3 THEN column_j
      WHEN i=4 THEN column_g
      WHEN i=5 THEN column_k
      ELSE column_l
    END
FROM table_1
ORDER BY 1;
```

In the following query, the CASE expression returns a VARCHAR result because the THEN and ELSE clause contains FLOAT and VARCHAR values. The length of the result is 30 since the default format for FLOAT is a string less than 30 characters, and USER is defined as VARCHAR(30) CHARACTER SET UNICODE. The result is CHARACTER SET UNICODE because USER is UNICODE.

```
SELECT a, CASE
      WHEN a=1
      THEN TIME
      ELSE USER
```

```

        END
FROM table_1
ORDER BY 1;

```

## Examples of Character Data in a CASE Expression: Example 2

The result of the following query is a 5354 failure (Arguments must be of type KANJI1) because one THEN/ELSE expression is a KANJI1 literal, but the server character sets of all the other THEN/ELSE expressions are not KANJI1.

```

SELECT i, CASE
    WHEN i=1 THEN column_l
    WHEN i=2 THEN column_u
    WHEN i=3 THEN column_j
    WHEN i=4 THEN column_g
    WHEN i=5 THEN _Kanji1'4142'XC
    ELSE column_k
END
FROM table_1
ORDER BY 1;

```

For this example, assume the following table definition:

```

CREATE table_1
(i          INTEGER,
 column_l  CHARACTER(10) CHARACTER SET LATIN,
 column_u  CHARACTER(10) CHARACTER SET UNICODE,
 column_j  CHARACTER(10) CHARACTER SET KANJISJIS,
 column_g  CHARACTER(10) CHARACTER SET GRAPHIC,
 column_k  CHARACTER(10) CHARACTER SET KANJI1);

```

The following query fails because the server character set is GRAPHIC (because the server character set of the first THEN with a character type is GRAPHIC):

```

SELECT i, CASE
    WHEN i=1 THEN 4
    WHEN i=2 THEN column_g
    WHEN i=3 THEN 5
    WHEN i=4 THEN column_l
    WHEN i=5 THEN column_k
    ELSE 10
END
FROM table_1
ORDER BY 1;

```

## Examples of Character Data in a CASE Expression: Example 3

One THEN/ELSE expression in the following query has a Unicode column. The query is successful and the result data type is UNICODE because the CASE expression contains 1 Unicode column and all other literals can be successfully translated to Unicode.

```
SELECT i, CASE
      WHEN i=1 THEN column_u
      WHEN i=2 THEN 'abc'
      WHEN i=3 THEN 8
      WHEN i=4 THEN _KanjiSJIS'4142'XC
      ELSE 10
    END
FROM table_1
ORDER BY 1;
```

## Examples of Character Data in a CASE Expression: Example 4

One THEN/ELSE expression in the following query has a Latin column. The query is successful and the result data type is Latin because the other literals can be successfully translated to Latin.

```
SELECT i, CASE
      WHEN i=1 THEN 'abc'
      WHEN i=2 THEN column_l
      ELSE 'def'
    END
FROM table_1
ORDER BY 1;
```

## THEN/ELSE Expressions Having Mixed Data Types

The rules for mixed data appear in the following table.

IF the THEN / ELSE clause expressions ...	THEN ...
consist of BYTE and/or VARBYTE data types	if the data types of all of the THEN/ELSE expressions are BYTE, the result data type is BYTE. If one or more expressions are VARBYTE, the result data type is VARBYTE. The result has a length equal to the maximum length of the different byte data types.
contain a DateTime or Interval data type	all of the THEN/ELSE clause expressions must have the same data type.

IF the THEN / ELSE clause expressions ...	THEN ...
contain a FLOAT (approximate numeric) and no character strings	<p>the CASE expression returns a FLOAT result.</p> <p><b>Note:</b> Some inaccuracy is inherent and unavoidable when FLOAT data types are involved.</p>
are composed only of DECIMAL data	<p>the CASE expression returns a DECIMAL result.</p> <p><b>Note:</b> A DECIMAL arithmetic result can have up to 38 digits. A result larger than 38 digits produces a numeric overflow error. For information about how the precision and scale of DECIMAL results are calculated, see <a href="#">Binary Arithmetic Result Data Types</a>. all are implicitly converted to FLOAT and the CASE expression returns a FLOAT result.</p> <p><b>Note:</b> Some inaccuracy is inherent and unavoidable when FLOAT data types are involved. Implicit conversion of DECIMAL and INTEGER values to FLOAT values may result in a loss of precision or produce a number that cannot be represented exactly.</p>
are composed only of mixed DECIMAL, BYTEINT, SMALLINT, INTEGER, and BIGINT data	<p>the CASE expression returns a DECIMAL result.</p> <p><b>Note:</b> A DECIMAL arithmetic result can have up to 38 digits. A result larger than 38 digits produces a numeric overflow error. For information about how the precision and scale of DECIMAL results are calculated, see <a href="#">Binary Arithmetic Result Data Types</a>. all are implicitly converted to FLOAT and the CASE expression returns a FLOAT result.</p> <p><b>Note:</b> Some inaccuracy is inherent and unavoidable when FLOAT data types are involved. Implicit conversion of DECIMAL and INTEGER values to FLOAT values may result in a loss of precision or produce a number that cannot be represented exactly.</p>
are a mix of BYTEINT, SMALLINT, INTEGER, and BIGINT data	<p>the resulting type is the largest type of any of the THEN/ELSE clause expressions, where the following list orders the types from largest to smallest:</p> <ul style="list-style-type: none"> <li>• BIGINT</li> <li>• INTEGER</li> <li>• SMALLINT</li> <li>• BYTEINT</li> </ul>
are composed only of numeric and character data	<p>the numeric data is converted to CHARACTER with a length as determined by the format associated with the numeric expression. Then, the rules for the result data type for character, length, and character set are applied. For details, see <a href="#">THEN/ELSE Character Type Expressions</a>.</p> <p><b>Note:</b> An error is generated if the server character set is GRAPHIC.</p>

## Examples of Numeric Data in a CASE Expression

For the following examples of numeric data behavior, assume the following table definitions for the CASE examples:

```
CREATE TABLE dec22
(column_1 INTEGER
,column_2 INTEGER
,column_3 DECIMAL(22,2) );
```

## Example: CASE Expression Fails

In the following statement, the CASE expression fails when column\_2 contains the value 1 and column\_3 contains the value 11223344556677889900.12 because the result is a DECIMAL value that requires more than 38 digits of precision:

```
SELECT SUM (CASE
            WHEN column_2=1
            THEN column_3 * 6.1122334455667788000000
            ELSE column_3
            END )
FROM dec22;
```

## Example: Shortening the Scale of the Multiplier

The following query corrects the problem in Example: CASE Expression Fails by shortening the scale of the multiplier in the THEN expression:

```
SELECT SUM (CASE
            WHEN column_2=1
            THEN column_3 * 6.1122334455667788
            ELSE column_3
            END )
FROM dec22;
```

## Example: Returning a DECIMAL(38,2) Result

In the following query, the CASE expression returns a DECIMAL(38,2) result because the THEN and ELSE clauses contain DECIMAL values:

```
SELECT SUM (CASE
            WHEN column_2=1
            THEN column_3 * 6
            ELSE column_3
            END )
FROM dec22;
```

## Examples of Character and Numeric Data in a CASE Expression

The following examples illustrate the behavior of queries containing CASE expressions with a THEN/ELSE clause composed of numeric and character data.

## Examples of Character and Numeric Data in a CASE Expression: Example 1

In the following query, the CASE expression returns a VARCHAR result because the THEN and ELSE clause contains FLOAT and VARCHAR values. The length of the result is 30 since the default format for FLOAT is a string less than 30 characters, and USER is defined as VARCHAR(30) CHARACTER SET UNICODE. The result is CHARACTER SET UNICODE because USER is UNICODE.

```
SELECT a, CASE
        WHEN a=1
        THEN TIME
        ELSE USER
      END
FROM table_1
ORDER BY 1;
```

## Examples of Character and Numeric Data in a CASE Expression: Example 2

For this example, assume the following table definition:

```
CREATE table_1
(i          INTEGER,
 column_l  CHARACTER(10) CHARACTER SET LATIN,
 column_u  CHARACTER(10) CHARACTER SET UNICODE,
 column_j  CHARACTER(10) CHARACTER SET KANJISJIS,
 column_g  CHARACTER(10) CHARACTER SET GRAPHIC,
 column_k  CHARACTER(10) CHARACTER SET KANJI1);
```

The following query fails because the server character set is GRAPHIC (because the server character set of the first THEN with a character type is GRAPHIC):

```
SELECT i, CASE
        WHEN i=1 THEN 4
        WHEN i=2 THEN column_g
        WHEN i=3 THEN 5
        WHEN i=4 THEN column_l
        WHEN i=5 THEN column_k
        ELSE 10
      END
FROM table_1
ORDER BY 1;
```

## Related Topics

For more information, see:

- [Binary Arithmetic Result Data Types](#)

## Format for a CASE Expression

### Default Format

The result of a CASE expression is displayed using the default format for the resulting data type. The result of a CASE expression does not apply the explicit format that may be defined for a column appearing in a THEN/ELSE expression.

Consider the following table definition:

```
CREATE TABLE duration
  (i INTEGER
   ,start_date DATE FORMAT 'EEEEBMMBDD,BYYYY'
   ,end_date DATE FORMAT 'DDBM3BY4' );
```

Assume the default format for the DATE data type is 'YY/MM/DD'.

The following query displays the result of the CASE expression using the 'YY/MM/DD' default DATE format, not the format defined for the *start\_date* or *end\_date* columns:

```
SELECT i, CASE
      WHEN i=1
      THEN start_date
      WHEN i=2
      THEN end_date
    END
FROM duration
ORDER BY 1;
```

## Using Explicit Type Conversion to Change Format

To modify the format of the result of a CASE expression, use CAST and specify the FORMAT clause.

Here is an example that uses CAST to change the format of the result of the CASE expression in the previous query.

```
SELECT i, ( CAST ((CASE
      WHEN i=1
      THEN start_date
      WHEN i=2
```

```

        THEN end_date
      END) AS DATE FORMAT 'M4BDD,BYYYY'))
FROM duration
ORDER BY 1;

```

For information on the default data type formats and the FORMAT phrase, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

## CASE and Nulls

The ANSI SQL:2011 standard specifies that the CASE expression and its related expressions COALESCE and NULLIF must be capable of returning a null result.

### Nulls and CASE Expressions

The rules for null usage in CASE, NULLIF, and COALESCE expressions are as follows.

- If no ELSE clause is specified in a CASE expression and the evaluation falls through all the WHEN clauses, the result is null.
- Nulls and expressions containing nulls are valid as *value\_expression\_1* in a valued CASE expression.

The following examples are valid.

```

SELECT CASE NULL
      WHEN 10
      THEN 'TEN'
      END;

SELECT CASE NULL + 1
      WHEN 10
      THEN 'TEN'
      END;

```

Both of the preceding examples return NULL because no ELSE clause is specified, and the evaluation falls through the WHEN clause because NULL is not equal to any value or to NULL.

- Comparing NULL to any value or to NULL is always FALSE. When testing for NULL, it is best to use a searched CASE expression using IS NULL or IS NOT NULL in the WHEN condition.

The following example is valid.

```

SELECT CASE
      WHEN column_1 IS NULL
      THEN 'NULL'
      END
FROM table_1;

```

Often, Teradata Database can detect when an expression that always evaluates to NULL is compared to some other expression or NULL, and gives an error that recommends using IS NULL or IS NOT NULL instead. Note that ANSI SQL does not consider this to be an error; however, Teradata Database reports an error since it is unlikely that comparing NULL in this manner is the intent of the user.

The following examples are not legal.

```
SELECT CASE column_1
      WHEN NULL
      THEN 'NULL'
      END
FROM table_1;

SELECT CASE column_1
      WHEN NULL + 1
      THEN 'NULL'
      END
FROM table_1;
SELECT CASE
      WHEN column_1 = NULL
      THEN 'NULL'
      END
FROM table_1;
SELECT CASE
      WHEN column_1 = NULL + 1
      THEN 'NULL'
      END
FROM table_1;
```

- Nulls and expressions containing nulls are valid as THEN clause expressions.

The following example is valid.

```
SELECT CASE
      WHEN column_1 = 10
      THEN NULL
      END
FROM table_1
```

Note that, unlike the previous examples, the NULL in the THEN clause is an SQL keyword and not the value of a character literal.

## CASE Shorthands

ANSI also defines two shorthand special cases of CASE specifically for handling nulls.

- COALESCE expression (see [COALESCE Expression](#))

- NULLIF expression (see [NULLIF Expression](#))

## COALESCE Expression

### Purpose

Returns NULL if all its arguments evaluate to null. Otherwise, it returns the value of the first non-null argument in the *scalar\_expression* list.

COALESCE is a shorthand expression for the following full CASE expression:

```
CASE
  WHEN  scalar_expression_1  IS NOT NULL
  THEN  scalar_expression_1
  ...
  WHEN  scalar_expression_n  IS NOT NULL
  THEN  scalar_expression_n
  ELSE  NULL
END
```

### Syntax

— COALESCE — ( — *scalar\_expression\_n* — ) —

## Syntax Elements

*scalar\_expression\_n*

An argument list.

Each COALESCE function must have at least two operands.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Usage Notes

A *scalar\_expression\_n* in the argument list may be evaluated twice: once as a search condition and again as a return value for that search condition.

Using a nondeterministic function, such as `RANDOM`, in a *scalar\_expression\_n* may have unexpected results, because if the first calculation of *scalar\_expression\_n* is not `NULL`, the second calculation of that *scalar\_expression\_n*, which is returned as the value of the `COALESCE` expression, might be `NULL`.

You can use a scalar subquery in a `COALESCE` expression. However, if you use a non-scalar subquery (a subquery that returns more than one row), a runtime error is returned.

## Default Title

The default title for a `COALESCE` expression appears as:

```
<CASE expression>
```

## Restrictions on the Data Types in a COALESCE Expression

The following restrictions apply to `CLOB`, `BLOB`, and `UDT` types in a `COALESCE` expression.

Data Type	Restrictions
<code>BLOB</code>	A <code>BLOB</code> can only appear in the argument list when it is cast to <code>BYTE</code> or <code>VARBYTE</code> .
<code>CLOB</code>	A <code>CLOB</code> can only appear in the argument list when it is cast to <code>CHAR</code> or <code>VARCHAR</code> .
<code>UDT</code>	Multiple <code>UDTs</code> can appear in the argument list only when they are identical types because Teradata Database does not perform implicit type conversion on <code>UDTs</code> in a <code>COALESCE</code> expression.

## Examples

### Example: Querying for a Phone Number

The following example returns the home phone number of the named individual (if present), or office phone if `HomePhone` is null, or `MessageService` if present and both home and office phone values are null. Returns `NULL` if all three values are null.

```
SELECT Name, COALESCE (HomePhone, OfficePhone, MessageService)
FROM PhoneDir;
```

### Example: Using COALESCE with an Arithmetic Operator

The following example uses `COALESCE` with an arithmetic operator.

```
SELECT COALESCE(Boxes,0) * 100
FROM Shipments;
```

## Example: Using COALESCE with an Comparison Operator

The following example uses COALESCE with a comparison operator.

```
SELECT Name
FROM Directory
WHERE Organization <> COALESCE (Level1, Level2, Level3);
```

## Related Topics

For more information, see:

- For additional information, such as the rules for evaluation and result data type, see [CASE](#).

## NULLIF Expression

### Purpose

Returns NULL if its arguments are equal. Otherwise, it returns its first argument, *scalar\_expression\_1*.

NULLIF is a shorthand expression for the following full CASE expression:

```
CASE
  WHEN  scalar_expression_1=scalar_expression_2
  THEN NULL
  ELSE  scalar_expression_1
END
```

### Syntax

— NULLIF — ( — *scalar\_expression1*, *scalar\_expression2* — ) —

## Syntax Elements

*scalar\_expression\_1*

The scalar expression to the left of the = in the expanded CASE expression, as shown in [Purpose](#).

*scalar\_expression\_2*

The scalar expression to the right of the = in the expanded CASE expression, as shown in [Purpose](#).

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Usage Notes

The *scalar\_expression\_1* argument may be evaluated twice: once as part of the search condition (see the preceding expanded CASE expression) and again as a return value for the ELSE clause.

Using a nondeterministic function, such as RANDOM, may have unexpected results if the first calculation of *scalar\_expression\_1* is not equal to *scalar\_expression\_2*, in which case the result of the CASE expression is the value of the second calculation of *scalar\_expression\_1*, which may be equal to *scalar\_expression\_2*.

You can use a scalar subquery in a NULLIF expression. However, if you use a non-scalar subquery (a subquery that returns more than one row), a runtime error is returned.

## Default Title

The default title for a NULLIF expression appears as:

```
<CASE expression>
```

## Restrictions on the Data Types in a NULLIF Expression

The following restrictions apply to CLOB, BLOB, and UDT types in a NULLIF expression.

Data Type	Restrictions
BLOB	A BLOB can only appear in the argument list when it is cast to BYTE or VARBYTE.
CLOB	A CLOB can only appear in the argument list when it is cast to CHAR or VARCHAR.
UDT	Multiple UDTs can appear in the argument list only when they are identical types and have an ordering definition.

## Examples

The following examples show queries on the following table:

```
CREATE TABLE Membership
  (FullName CHARACTER(39)
  ,Age SMALLINT
  ,Code CHARACTER(4) );
```

**Example: Querying with the ANSI-Compliant Form**

Here is the ANSI-compliant form of the Teradata SQL NULLIFZERO(Age) function, and is more versatile.

```
SELECT FullName, NULLIF (Age,0) FROM Membership;
```

**Example: Blank Spaces**

In the following query, blanks indicate no value.

```
SELECT FullName, NULLIF (Code, ' ') FROM Membership;
```

**Example: Querying for NULLIF in an Expression with an Arithmetic Operator**

The following example uses NULLIF in an expression with an arithmetic operator.

```
SELECT NULLIF(Age,0) * 100;
```

**Related Topics**

For more information, see:

- For additional information, such as the rules for evaluation and result data type, see [CASE](#).

## Module 5: CASE Expressions

Upon completion of this module, you should be able to:

- Return alternate values using the **CASE** expression.
- Use special variations of the **CASE** expression:
  - **NULLIF**
  - **COALESCE**

The **CASE** expression allows for conditional processing of returned rows. It provides some IF-THEN-ELSE logic on top of the normally set-based SQL.

There are two general categories of **CASE** expressions:

- **Valued CASE** expressions
- **Searched CASE** expressions

**Valued case** statements have simpler syntax, but they are limited. **Searched case** statements have more complex syntax, but more flexibility.

In this course, we will look at valued case statements first.

A **valued CASE expression** looks like this:

```
CASE value-expr WHEN expr1 THEN result1
      WHEN expr2 THEN result2
      :
      ELSE resultn
END
```

**Value-expr** is either a column or an expression involving columns. For each row, Teradata database will compare **value-expr** with **exprn** values until there is a match:

1. If **value-expr** equals **expr1**, output **result1**. If they are not equal, go to step 2.
2. If **value-expr** equals **expr2**, output **result2**. If they are not equal, go to step 3.
3. If **value-expr** equals **expr3**, output **result3**. If they are not equal, go to step 4...

In this way, Teradata database will test all **WHEN** clauses until we get a match.

If there is no match, then we will use the **ELSE** condition and output **resultn**.

## Example

What fraction of the total salary of all employees is the sum of salaries from department 401?

```
SELECT SUM(CASE department_number
            WHEN 401 THEN salary_amount
            ELSE 0
          END) / SUM(salary_amount)
FROM employee;
```

(Sum(<CASE expression>)/Sum(salary_amount))
0.22

### Things to Notice:

- The case statement says that if the department number is 401, then sum the salary amount. If it is anything else, then sum a 0. This will get only the sum of salaries from department 401.
- To get the fraction of the total salary, we have a denominator that simply sums all the salary amounts.
- The default output title is lengthy. You can use the **AS** clause to give it a better name.

```
SELECT SUM(CASE department_number
            WHEN 401 THEN salary_amount
            ELSE 0
          END) / SUM(salary_amount) AS sal_ratio
FROM employee;
```

Here is another example of a valued case statement. From a performance standpoint, this might be better to answer with a filter instead of a CASE statement, but this will still give you the correct answer.

### Example

Get the total of salaries for departments 401 and 501.

```
SELECT SUM(CASE department_number
            WHEN 401 THEN salary_amount
            WHEN 501 THEN salary_amount
            ELSE 0
            END) AS total_sals_401_501
FROM employee;
```

total_sals_401_501
--------------------

445700.00
-----------

### Things to Notice:

- The case statement says that if the department number is 401, then sum the salary amount. If it is anything else, then sum a 0. This will get only the sum of salaries from department 401.
- To get the fraction of the total salary, we have a denominator that simply sums all the salary amounts.
- The default output title is lengthy. You can use the **AS** clause to give it a better name.

The searched case syntax is more complex, but offers more functionality than the valued CASE statement. A **searched CASE expression** looks like this:

```
CASE WHEN condition1 THEN value-expr1
      WHEN condition2 THEN value-expr2
      :
      ELSE value-exprn
END
```

For each row, Teradata database will evaluate the conditions in order until there is a hit. If there are no conditions met, then we will use the **ELSE** condition and output **value-exprn**. This is similar to the valued CASE expression.

However, instead of a simple evaluation for equality like a valued CASE expression, the **conditions** within a **searched CASE expression** allow you much greater flexibility:

- Each condition may involve equality or non-equality operators.
- Each condition may involve multiple columns.
- **Condition1** and **condition2** may involve different columns.

In this example, you can see the flexibility provided by the searched case expression.

### Example

Place each employee in a salary category: under \$30K, under \$40K, under \$50K, and anything above \$50K.

```
SELECT last_name,
       CASE WHEN salary_amount < 30000 THEN 'Under $30K'
            WHEN salary_amount < 40000 THEN 'Under $40K'
            WHEN salary_amount < 50000 THEN 'Under $50K'
            ELSE '> = $50K'
       END
FROM employee
ORDER BY salary_amount;
```

last_name	salary_category
Crane	Under \$30K
Phillips	Under \$30K
:	:
Lombardo	Under \$40K
Ryan	Under \$40K
:	:
Brown	Under \$50K
Brown	Under \$50K
:	:
Daly	> = \$50K
Wilson	> = \$50K
:	:

### Things to Notice:

We are not restricted to an equality condition; in this case we use a less than (<) comparison.

## Example

Calculate the fraction of the total salaries represented by departments 401 and 501. Allow for a 10% salary increase for employees in department 501.

```
SELECT SUM (CASE WHEN department_number = 401 THEN salary_amount
                WHEN department_number = 501 THEN salary_amount * 1.1
                ELSE 0
            END) /
        SUM (CASE WHEN department_number = 501 THEN salary_amount * 1.1
                ELSE salary_amount
            END ) AS sal_ratio
FROM employee;
```

sal_ratio
0.415

### Things to Notice:

The numerator CASE calculates the sum of salaries for departments 401 and 501, including a 10% increase for 501 employees.

The denominator CASE calculates the sum of all salaries for all departments including a 10% increase for 501 employees.

## Example

Find the people who qualify for early retirement and which plan they qualify for.

Plan	Age	Years Service
Gold	Over 60	Over 20
Silver	Over 55	Over 15
Bronze	Over 50	Over 10

```

SELECT  CAST(last_name AS CHAR(15))
        , (CURRENT_DATE - hire_date)/365.25 AS On_The_Job
        , (CURRENT_DATE - Birthdate)/365.25 AS Age
        , CASE WHEN Age > 60 AND On_The_Job > 20 THEN 'Gold Plan'
              WHEN Age > 55 AND On_The_Job > 15 THEN 'Silver Plan'
              ELSE 'Bronze Plan'
        END AS Plan
FROM employee
WHERE Age > 50 AND On_The_Job > 10
ORDER BY 4 DESC;

```

Results will vary depending on when the query is executed.

### Things to Notice:

A complex searched CASE expression may involve tests on multiple columns under multiple conditions. The **WHERE** clause limits the number of rows to be returned, while the **CASE** statement determines the disposition of the qualifying rows.

**NULLIF** returns NULL if its arguments are equal. Otherwise, it returns its first argument, *scalar\_expression\_1*.

```
NULLIF ( scalar_expression1, scalar_expression2 )
```

**NULLIF** is a special expression that is shorthand for the following full CASE expression:

```
CASE WHEN scalar_expression_1 = scalar_expression_2 THEN NULL  
      ELSE scalar_expression_1  
END
```

We will use the below **CALL\_EMPLOYEE** table in the next few examples. Note especially the **labor\_hours** column, which has 3 values (one of which is 0) and 3 nulls.

call_number	employee_number	call_status_code	assigned_date	assigned_time	finished_date	finished_time
5	1004	5	1161216	1025		
4	1010	1	1161215	1250		
1	1004	1	1161215	0905	1161216	1625
6	1004	2	1161216	1110		
3	1001	16	1161215	1215		
2	1001	2	1161215	0930	1161216	1375

**NULLIF** can transform a zero (0) into a null. It is the ANSI standard substitute for Teradata database's **NULLIFZERO** function.

### Example

```
SELECT call_number
       ,labor_hours  (TITLE 'ACTUAL HOURS')
       ,NULLIF (labor_hours, 0) (TITLE 'NULLIF ZERO HOURS')
FROM   call_employee
ORDER BY labor_hours;
```

call_number	ACTUAL HOURS	NULLIF ZERO HOURS
4	null	null
5	null	null
6	null	null
3	.0	null
2	4.0	4.0
1	8.5	8.5

### Things to Notice

Call\_number 3 has zero labor\_hours.

The third column shows 0 transformed to null through use of **NULLIF** function.

A common use for **NULLIF** is in a denominator or divisor. Dividing by zero will abort a query and produce an error message, but dividing by a null will simply result in a null.

### Example

Find the ratio of hourly billing rate to hourly cost rate for all "analyst" jobs.

#### Without NULLIF:

```
SELECT description
      , hourly_billing_rate/hourly_cost_rate
        AS "Billing-Cost Ratio"
FROM   job
WHERE  description like '%analyst%' ;
```

Error Message: Division by zero in an expression involving  
job.hourly\_cost\_rate.

#### With NULLIF:

```
SELECT description
      , hourly_billing_rate /
        NULLIF(hourly_cost_rate, 0)
        AS "Billing-Cost Ratio"
FROM   job
WHERE  description like '%analyst%' ;
```

description	Billing-Cost Ratio
Software Analyst	1.29
System Support Analyst	null
System Analyst	1.14

Applying the **NULLIF** function to the denominator produces a null result. This avoids the error and allows a report to be generated.

**COALESCE** returns the value of the first non-null argument in the scalar\_expression list. If all of its arguments evaluate to null, it returns NULL.

```
COALESCE ( scalar_expression_1, scalar_expression_2, ... ,  
scalar_expression_n )
```

**COALESCE** is a special expression that is shorthand for the following full CASE expression:

```
CASE WHEN scalar_expression_1 IS NOT NULL THEN scalar_expression_1  
...  
      WHEN scalar_expression_n IS NOT NULL THEN scalar_expression_n  
      ELSE NULL  
END
```

### Example

Show office phone number if present, else show home phone.

```
SELECT name  
      ,COALESCE (office_phone, home_phone)  
FROM   phone_table;
```

**COALESCE** can be used to convert a possible NULL value to zero. **COALESCE** is the ANSI standard alternative for the Teradata database **ZEROIFNULL** function.

### Example

Get the number of students in each course. If *num\_students* is null, return a zero.

```
SELECT course_name
       ,COALESCE (num_students, 0) (TITLE '# Students')
FROM   class_schedule;
```

course_name	# Students
Teradata SQL	17
Physical DB Design	0

**COALESCE** can also be used to convert possible NULL value to a string literal 'NULL VALUE'. With **COALESCE**, you can change a null into whatever you want.

### Example

```
SELECT course_name
       ,COALESCE (num_students, 'NULL VALUE') (TITLE '# Students')
FROM   class_schedule;
```

course_name	# Students
Teradata SQL	17
Physical DB Design	NULL VALUE

The default is to ignore nulls in aggregations. **NULLIF** and **COALESCE** allow you to change that rule. In the example below (based on the [call\\_employee table we saw earlier](#)), you can see how you can manipulate the same data to get very different results with a simple aggregation. Which should you use? That depends on the meaning of the data, and what you are trying to show.

### Example

```
SELECT AVG (labor_hours)                (TITLE 'Default AVG')
      , AVG (NULLIF ( labor_hours, 0))  (TITLE 'NullIfZero AVG')
      , AVG (COALESCE (labor_hours, 0)) (TITLE 'ZeroIfNull AVG')
      , COUNT (labor_hours)              (TITLE 'Default COUNT')
      , COUNT (NULLIF (labor_hours, 0))  (TITLE 'NullIfZero COUNT')
      , COUNT (COALESCE (labor_hours, 0)) (TITLE 'ZeroIfNull COUNT')
FROM   call_employee;
```

call_number	labor_hours
5	
4	
1	8.5
6	
3	.0
2	4.0

Columns from CALL\_EMPLOYEE

Default AVG	NullIfZero AVG	ZeroIfNull AVG	Default COUNT	NullIfZero COUNT	ZeroIfNull COUNT
4.2	6.2	2.1	3	2	6

If you have not set up your lab server connection, click on the Lab Setup button at the bottom of the page to get instructions. You will need these instructions to log on to the Teradata database. If you experience problems connecting to the lab server, contact [Training.Support@Teradata.com](mailto:Training.Support@Teradata.com).

For this set of lab exercises you may need information from the [Database Info](#) document. Prior to doing these labs, it will be helpful to reset your default database to the **CustomerService** database (i.e. `DATABASE CustomerService;`).

Click on the Next button at the bottom of the page to see the answers.

- 1.) Calculate the fraction of the total company budget represented by departments 401 and 403.
- 2.) Calculate the fraction of the total company budget represented by departments 401 and 403 after department 403 has been given a 5% budget increase.
- 3.) Create a budget report from the **department** table. Show the total, average, minimum and maximum budget amounts. Title the columns "Total", "Avg", "Min" and "Max". Do the query twice:
  - a.) once treating NULL values as zero,
  - b.) once excluding NULL values in aggregates.

Compare the results.

- 4.) Accounting wants to find out which way of slanting the statistics is most beneficial to the company. Do a departmental salary list of the total salaries for each department, and the average departmental salary three times:
  - without any changes (treating NULLs as null and zero values as zero)
  - treating NULL values as zero
  - treating zero values as NULL

Also list the count of rows that are going into each averaging function. Sequence the report by department.

Title your columns as follows: Dept, Tot #, Tot Sal, Avg Sal, ZIN #, ZIN Avg, NIZ #, NIZ Avg

Are there differences in these computations? Why or why not?

**Solution 1**

```
SELECT SUM(CASE department_number WHEN 401 THEN budget_amount
                                   WHEN 403 THEN budget_amount
                                   ELSE 0
                                   END) / SUM(budget_amount) AS "401/403 Bgt Ratio"
FROM department;
```

```
401/403 Bgt Ratio
-----
.49
```

**Solution 2**

```
SELECT SUM(CASE WHEN department_number = 401 THEN budget_amount
                WHEN department_number = 403 THEN budget_amount * 1.05
                ELSE 0
                END) /
SUM(CASE WHEN department_number = 403 THEN budget_amount * 1.05
        ELSE budget_amount
        END ) AS Dept_401_403_Bgt_Ratio
FROM department;
```

```
Dept_401_403_Bgt_Ratio
-----
.4949
```

**Solution 3**

This solution is for option a) treating NULL values as zero

```
SELECT SUM (COALESCE (budget_amount,0)) (DEC(9,2)) AS "Total"
      ,AVG (COALESCE (budget_amount,0)) (DEC(9,2)) AS "Avg"
      ,MIN (COALESCE (budget_amount,0)) (DEC(9,2)) AS "Min"
      ,MAX (COALESCE (budget_amount,0)) (DEC(9,2)) AS "Max"
FROM department;
```

Total	Avg	Min	Max
3915700.00	435077.78	.00	982300.00

This solution is for option b) not including NULL values in aggregates.

```
SELECT SUM(budget_amount) AS "Total"
      ,AVG(budget_amount) AS "Avg"
      ,MIN(budget_amount) AS "Min"
      ,MAX(budget_amount) AS "Max"
FROM department;
```

Total	Avg	Min	Max
3915700.00	489462.50	226000.00	982300.00

**Solution 4**

```

SELECT  department_number      AS "Dept"
        ,COUNT(salary_amount) AS "Tot #"
        ,SUM(salary_amount)    AS "Tot Sal"
        ,AVG(salary_amount)    AS "Avg Sal"
        ,COUNT(COALESCE(salary_amount,0)) AS "ZIN #"
        ,AVG(COALESCE(salary_amount,0))    AS "ZIN Avg"
        ,COUNT(NULLIF(salary_amount,0))   AS "NIZ #"
        ,AVG(NULLIF(salary_amount,0))      AS "NIZ Avg"
FROM    employee
GROUP BY 1
ORDER BY 1;

```

Dept	Tot #	Tot Sal	Avg Sal	ZIN #	ZIN Avg	NIZ #	NIZ Avg
----	-----	-----	-----	-----	-----	-----	-----
100	1	100000.00	100000.00	1	100000.00	1	100000.00
201	2	73450.00	36725.00	2	36725.00	2	36725.00
301	3	116400.00	38800.00	3	38800.00	3	38800.00
302	1	56500.00	56500.00	1	56500.00	1	56500.00
401	7	245575.00	35082.14	7	35082.14	7	35082.14
402	2	77000.00	38500.00	2	38500.00	2	38500.00
403	6	233000.00	38833.33	6	38833.33	6	38833.33
501	4	200125.00	50031.25	4	50031.25	4	50031.25

The Teradata logo is displayed in white text on a teal background that features a stylized, cloudy or nebula-like pattern. This graphic occupies the left portion of the slide.

teradata.

**1.10 - Given a scenario  
including the need to connect  
to an external data source,  
identify the SQL code snippet  
that should be used.**

Teradata Vantage Analytics Certification:  
Learning Resource

## Explaining the Query

- Query initiated from IDW
- Local query on IDW run to select qualifying rows; **sales\_quantity aggregated**
- Remote query on 1700 run to select qualifying rows; **sales\_quantity aggregated**
- **Qualifying rows** returned from the 1700 and placed in spool on IDW
- IDW merges both data sets
- IDW applies ordering

```
SELECT sales_date, SUM(sales_quantity) AS  
total_sales  
FROM samples.sales_fact  
GROUP BY 1  
UNION ALL  
SELECT *  
FROM FOREIGN TABLE (  
SELECT sales_date, SUM(sales_quantity)  
AS total_sales  
FROM samples.sales_fact_history  
GROUP BY 1)@ td1700 old_sales  
ORDER BY 1;
```

Query Result:	1,336 rows	1,336 rows
Rows transferred:	1,002	14 Million
Elapsed time:	~4 sec	~30 sec

teradata.

## QueryGrid 2.0

- Define: Foreign Server, Catalog Properties, Storage Handlers

### QG 2.0 Foreign Server Definition

```
CREATE FOREIGN SERVER hdp USING  
LINK ('TD2P')  
DO IMPORT WITH TD_SYSFNLIB.QGINITIATORIMPORT ,  
DO EXPORT WITH TD_SYSFNLIB.QGINITIATOREXPORT ;
```

Remaining name value  
pairs obtained from QGM  
from link name

Teradata query joining current data with archive data in Presto/Hadoop

```
SELECT * FROM websales_current UNION ALL SELECT * FROM websales_archive@hdp;
```

Defined Using  
Foreign Server

3

teradata.

- The link configured in the QGM will be used to create the foreign server object in Teradata, this object enables the @foreign\_server name remote queries on Teradata
- Contrast the old FS with the new FS – simplicity.

## QueryGrid 2.0

- Define: Foreign Server, Catalog Properties, Storage Handlers

### QG 2.0 Presto Catalog Properties

```
connector.name=gginitiator  
gginitiator.linkName=P2TD  
gginitiator.version=active
```

Remaining name value  
pairs obtained from QGM  
from link name

Presto query joining current data in Teradata with archive data in Presto/Hadoop

```
SELECT * FROM td.sales.websales_current UNION ALL SELECT * FROM  
hive.sales.websales_archive;
```

Defined Using  
Catalog Properties

4

teradata.

- If your use case requires that the query to initiate from Presto, then create a link in the QGM (P2TD) then use it to create a catalog (name: td) for Teradata, the catalog is referenced in a remote query as in this example

## QueryGrid 2.0

- Define: Foreign Server, Catalog Properties, Storage Handlers

### QG 2.0 Hive Storage Handler Definition

```
CREATE TABLE websales_current  
ROW FORMAT SERDE 'com.teradata.querygrid.qgc.hive.QGSerDe'  
STORED BY 'com.teradata.querygrid.qgc.hive.QGStorageHandler'  
TBLPROPERTIES ("link"="H2TD",  
"table"="sales.websales_current");
```

Remaining name value  
pairs obtained from QGM  
from link name

Hive query joining current data in Teradata with archive data in Hive/Hadoop

```
SELECT * FROM websales_current UNION ALL SELECT * FROM websales_archive;
```

Defined Using  
Storage Handlers

- If your use case requires that the query to initiate from Hive, then create a link in the QGM (H2TD) then use it to create a storage handler for every table you want to access remotely, the storage handler is referenced in a remote query as in this example

**Thank you.**

**teradata.**

©2018 Teradata



## Module 16: Permanent and Derived Tables

Teradata SQL for Business Users

Copyright © 2007–2020 by Teradata. All Rights Reserved.

## Permanent and Derived Tables

After completing this module, you will be able to:

- Distinguish between SET and MULTiset tables.
- Identify table level options in a CREATE TABLE.
- Identify column level options in a CREATE TABLE.
- Identify index level options in a CREATE TABLE.
- Create and drop secondary indexes on existing tables.
- Distinguish between deleting tables and dropping tables.
- Return help information for a table's indexes.
- Use permanent tables for temporary use.
- Use derived tables for temporary use.
- Distinguish between the various forms used to define a derived table.
- Involve derived tables in multiple table joins.

## Data Definition Language

CREATE TABLE is a DDL request.

Data Definition Language (DDL) is used by SQL to create, modify and remove object definitions.

These definitions are stored in the dictionary found in the user DBC.

Changes to dictionary tables require a write lock, and can block the database's attempts to access this locked information during parsing.

```
CREATE < SET/MULTISET > TABLE tablename, < Table Level Attributes >  
(   column name datatype < Column Level Attributes >  
    . . . )  
< Primary and Secondary Index Level Attributes >;
```

As stated on this slide, DDL requests are those which require updates to the dictionary. In those cases, the database must place a “write” lock on a dictionary table. This can be observed in an EXPLAIN of the request.

While the dictionary is write locked on an object (e.g., a table, a view, etc.), attempts by the parser to “resolve” the object for concurrent accesses block on the “write” lock, preventing those queries from being parsed until the DDL is finished. This is normally an issue with explicit transaction processing and not with implicit transaction processing.

## Table Level Options

Table options are those that affect the entire table. These options are defaulted to those of the database in which the table resides.

To see all of the default options for the database, type and submit the following:

→ `HELP 'SQL CREATE DATABASE';`

`SHOW TABLE Department;`

```
CREATE SET TABLE XYZ.department ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
(
    department_number SMALLINT,
    department_name CHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
    budget_amount DECIMAL(10,2),
    manager_employee_number INTEGER
)
UNIQUE PRIMARY INDEX ( department_number )
UNIQUE INDEX ( department_name );
```

These may typically be the default assignments.

The table level options are those that have an effect on the table as a whole. These options are better discussed in a course on Physical Design rather than in a SQL course. The following is a list of table level options, and can be obtained by issuing the following SQL command:

### HELP 'SQL CREATE TABLE'

```
{ CREATE [SET      ] [ GLOBAL TEMPORARY ] TABLE }
{ CREATE [MULTISET] [ VOLATILE ]          }
{                                          }
[ databasename. ] tablename
{ CT                                          }
```

```
[NO] QUEUE
[NO] FALLBACK [PROTECTION]
[NO] LOG

[NO ]
[    ] [BEFORE] JOURNAL
[DUAL]
```

```
[NO      ]
[DUAL    ]
```

```

[          ]          AFTER JOURNAL
[LOCAL    ]
[NOT LOCAL]

WITH JOURNAL TABLE = [databasename.]tablename

CHECKSUM = { DEFAULT | NONE | LOW | MEDIUM | HIGH | ALL
}
FREESPACE = integer [PERCENT]
{ DATABLOCKSIZE = integer [BYTES | KBYTES | KILOBYTES]
}
{
}
{ [MINIMUM | MAXIMUM] DATABLOCKSIZE
}

```

## Set vs. Multiset

The default for Teradata Mode tables is “SET” *(No duplicate rows allowed)*

The default for ANSI Mode tables is “MULTISET” *(Duplicate rows allowed)*

A duplicate row is where each and every column value for one row is equal to its corresponding column value in another row.

For values that are defined as “case sensitive,” uppercase values differ from corresponding lower case values for the same character value.

For values defined as “not case sensitive” equal character values are the same whether upper case or lowercase.

Dept#	Last Name	First Name
100	'Smith'	'Mary'
100	'Smith'	'Mary '
100	'smith'	'Mary'
100	'Smith'	'mary'

If both character columns are not case sensitive, which rows would be duplicates?

If both character columns are case sensitive, which rows would be duplicates?

*(Answers in the student manual.)*

The keywords MULTISET and SET describe whether-or-not duplicates are allowed respectively. These keywords appear in the CREATE TABLE as shown in the example.

**CREATE [SET | MULTISET] TABLE abc ...**

The defaults are SET (in Teradata mode) and MULTISET (in ANSI mode).

The concept of a duplicate row is a very important one in database theory, and case sensitivity plays an important role in this discussion. Since Teradata is, by default, not case sensitive, upper vs. lower case characters, for the same character, evaluate as being the same character value. While as with case specific-ness, upper vs. lower characters evaluate as being different for the same character value. The biggest concern between the two is beyond the scope of this class and has to deal with the following question:

**“Why allow duplicate rows at all?”**

As SQL goes, we simply discuss SET and MULTISET concepts and syntax, not strategies.

On this slide:

***If both character columns are not case sensitive, which rows would be duplicates?***

*Answer: all but the second row.*

***If both character columns are case sensitive, which rows would be duplicates?***

*Answer: only the second row.*

## CREATE TABLE

Table Protection Options (*blue indicate defaults*)

- **FALLBACK** or NO FALLBACK
- BEFORE JOURNAL (**NO**, SINGLE or DUAL)
- AFTER JOURNAL (**NO**, SINGLE (LOCAL or NOT LOCAL) or DUAL)
- WITH JOURNAL TABLE [*tablename*]
- CHECKSUM

Space Management Options

- **FREESPACE = *n* [PERCENT]** – specifies the percentage of space to remain free during a loading operation.
- **DATABLOCKSIZE** – specifies the maximum block size for multi-row data blocks. The data block is the unit of I/O in Teradata database.
- **MERGEBLOCKRATIO = *n* [PERCENT]** – provides a way to combine existing small data blocks into a single larger data block. *n* specifies the size of the single larger data block, as a percentage of the table's maximum block size. The system default is 60.
- **BLOCKCOMPRESSION** – sets the temperature-based block compression state.

### Protection Options

The CREATE TABLE statement allows you to specify **physical implementation options** such as fallback and permanent journal protection, data block size, and cylinder freespace factor. You may define a primary index (different than the primary key) as well as secondary indexes to improve physical access to data in the table.

**FALLBACK** specifies that the system builds a duplicate copy of each row of the table and stores it on a different (FALLBACK) AMP within the cluster.

**JOURNAL** specifies that the system stores **BEFORE** and/or **AFTER** images for each changed row of the table in the specified permanent journal, providing disaster recovery capability.

The **CHECKSUM** option implements disk I/O integrity checking of primary table data rows, fallback table data rows, and secondary index subtable rows at various user-specified or system-wide levels.

### Space Management Options

Example

```
CREATE SET TABLE table1, NO FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
```

CHECKSUM = DEFAULT,  
BLOCKCOMPRESSION = AUTOTEMP,  
FREESPACE = 10 PERCENT,  
DATABLOCKSIZE = 254  
(c1 INTEGER)  
UNIQUE PRIMARY INDEX (c1);

## Column Level Options

Columns may be assigned:

- a name
- a column attribute
- a data type
- a data type attribute
- to a constraint

For a more complete list  
see the student manual.

**SHOW TABLE Department;**

```
CREATE SET TABLE XYZ.department ,FALLBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
  department_number SMALLINT,
  department_name CHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
  budget_amount DECIMAL(10,2),
  manager_employee_number INTEGER,
  location_name VARCHAR(100)
)
UNIQUE PRIMARY INDEX ( department_number )
UNIQUE INDEX ( department_name );
```

Diagram illustrating column level options in the SQL command:

- Column Name:** department\_number, department\_name, budget\_amount, manager\_employee\_number, location\_name
- Column Data Type:** SMALLINT, CHAR(30), DECIMAL(10,2), INTEGER, VARCHAR(100)
- Column Data Type Attribute (defaulted assignment):** CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL

The column level options are those that have an effect on a table's columns as a whole. These options are better discussed in a course on Physical Design rather than in a SQL course. The following is a list of column level options only, and can be obtained by issuing the following SQL command:

### HELP 'SQL CREATE TABLE'

```
. . . . .
COLUMN_DECLARATION IS
  cname data_type_declaration [column_attribute [...,
                                          column_attribute]]

COLUMN_ATTRIBUTE IS ONE OF THE FOLLOWING:
COMPRESS [constant | ({NULL | constant} [..., {NULL | constant}])]
{ UNIQUE }
[CONSTRAINT name] { PRIMARY KEY }
{ CHECK (boolean_condition) }
{ references_option }

TABLE LEVEL OPTION IS ONE OF THE FOLLOWING:
[CONSTRAINT name] [UNIQUE | PRIMARY KEY] (cname[ ..., cname])
[CONSTRAINT name] FOREIGN KEY (cname [ ..., cname])
references_option
```

[CONSTRAINT name] CHECK (boolean condition)

REFERENCES\_OPTION IS

REFERENCES [WITH [NO] CHECK OPTION] tname [(cname [...,  
cname)]]

DATA TYPE ATTRIBUTES FOLLOW:

NOT NULL

UPPERCASE

[ NOT ] CASESPECIFIC

{ FORMAT | TITLE } quotestring

NAMED name

WITH DEFAULT character\_data\_type

{ number }

{ USER }

DEFAULT { DATE }

{ TIME }

{ NULL }

GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY

[(optional\_idcol\_parameters)]

## Index Level Options

For indexes:

- Only one primary index allowed per table.
- Up to 32 secondary indexes are allowed per table.
- Up to 64 columns are allowed per index.
- Indexes may be unique or non-unique.

For a more complete list  
see the student manual.

**SHOW TABLE Department;**

```
CREATE SET TABLE XYZ.department ,FALLBACK ,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT
(
  department_number SMALLINT,
  department_name CHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
  budget_amount DECIMAL(10,2),
  manager_employee_number INTEGER
)
UNIQUE PRIMARY INDEX ( department_number )
UNIQUE INDEX ( department_name )
INDEX ( manager_employee_number);
```

← Unique Primary Index  
← Unique Secondary Index  
← Non-Unique Secondary Index

The index level options are those that have an effect on the table's indexes as a whole. These options are better discussed in a course on Physical Design rather than in a SQL course. The following is a list of create table index options only, and can be obtained by issuing the following SQL command:

### HELP 'SQL CREATE TABLE'

. . . .

index is one of the following:

NO PRIMARY INDEX [partitioning]

[UNIQUE] PRIMARY INDEX [name] (cname [..., cname])  
[partitioning]

UNIQUE INDEX [name] (cname [..., cname])

INDEX [name] [ALL] (cname [..., cname]) [ORDER BY [VALUES |  
HASH] (cname)]

partitioning is one of the following:

```

PARTITION BY {general_expression |
              partitioning_expression |
              column_partition

PARTITION BY ( column_partitioning
              [, partitioning_expression [...,
              partitioning_expression]] )

PARTITION BY ( partitioning_expression [...,
              partitioning_expression]
              [, column_partitioning
              [, partitioning_expression [...,
              partitioning_expression]]]
.      .      .      .
}

```

It is possible to create a table without a primary index. To do this, simply use the following phrase in your table's DDL:

**No Primary Index;**

## Creating and Dropping Secondary Indexes

Secondary indexes may be created on existing tables.

- They may be defined as unique (USI) or non-unique (NUSI).
- They may be optionally named, or left unnamed.
- They may include up to 64 columns.

Named unique secondary index (USI) on employee name:

```
CREATE UNIQUE INDEX fullname (last_name, first_name) ON emp_data;  
DROP INDEX fullname ON emp_data;  
DROP INDEX (last_name, first_name) ON emp_data;
```

Unnamed, non-unique secondary index (NUSI) on job code:

```
CREATE INDEX (job_code) ON emp_data;  
DROP INDEX (job_code) ON emp_data;
```

Although secondary indexes may be defined within a CREATE TABLE, generally these are defined after the table has been created and populated with data. Once they are created, they may be repetitively dropped (e.g., prior to loading data from a file) and recreated.

Indexes may be provided with a name. This may be useful for dropping it later, by name. It may be easier, perhaps for some, to just simply replace the “CREATE” keyword (used when creating the index) with the “DROP” keyword. In this case, if the index is unique, do not provide the “UNIQUE” keyword. Examples are shown.

## Help Index

HELP INDEX emp\_data;

HELP INDEX shows information on all indexes defined for a table.

The values in the Index Id column correlate to the index numbers referenced in EXPLAIN text.

Unique?	Y
Primary//or//Secondary?	P
Column Names	employee_number
Index Id	1
Approximate Count	0
Index Name	Emp_Key
<hr/>	
Unique?	N
Primary//or//Secondary?	S
Column Names	department_number
Index Id	4
Approximate Count	0
Index Name	?
<hr/>	
Unique?	Y
Primary//or//Secondary?	S
Column Names	last_name,first_name
Index Id	8
Approximate Count	0
Index Name	Full_Name
<hr/>	
Unique?	N
Primary//or//Secondary?	S
Column Names	job_code
Index Id	12
Approximate Count	0
Index Name	?

The HELP INDEX command allows one to view information about table indexes. Except for the value for “approximate count,” each piece of information should be fairly straight forward. The approximate count value is obtained via a sampling of data and represents an approximate count of the number of distinct values of the sample. Normally this number should be fairly close to the number of unique values shown via a HELP STATS command. If they are very widely different values, this could be an indication that the statistics collected are stale (i.e., old) and need to be refreshed.

## Deleting vs. Dropping Tables

To remove all data associated with a table, without dropping the table definition from the Data Dictionary, use the DELETE statement.

Examples: `DELETE FROM emp_data ALL;`  
(all three are `DELETE FROM emp_data;`  
synonymous) `DELETE emp_data;`

- Deletes all data in emp\_data.
- Table definition remains in the Data Dictionary.
- Access rights remain unchanged.

To remove all data associated with a table, as well as the table structure definition from the Data Dictionary, use the DROP TABLE statement.

Example: `DROP TABLE emp_data;`

- Deletes all data in emp\_data.
- Removes table headers for emp\_data.
- Removes the emp\_data definition from the Data Dictionary.
- Removes all explicit access rights on the table.

The three delete commands at the top of this slide all perform equally fast. They each remove all of the rows from the table, leaving only the table definition. No transient journaling is performed on such a delete. Transient journaling provides a log of all of the changes taking place against a table during a transaction and provides for rollback capability for a failed transaction. This delete performs fast enough that it is unlikely to be able to abort once it begins. Having said this, if one could abort it in time, a rollback would occur without issue.

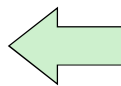
The drop command, at the bottom of this slide, removes not only the rows, but the table definition as well. It performs slightly slower than the preceding delete commands because of the amount of additional time that it takes to remove the dictionary entries.

## Using Real Tables – “Temporarily”

- The strategy below can be used to return the result for the business concern shown.
- Correlated Subqueries can return the result, but without projecting the average salary!
- Both queries are cross joins because no equality condition exists for the join condition.
- This table will eventually have to be dropped.

*Find employees whose salaries are greater than their department average.*

```
CREATE TABLE deptsal
( avgsal DEC(10,2) )
UNIQUE PRIMARY INDEX (avgsal)
```



```
INSERT INTO deptsal
SELECT AVG(salary_amount)
FROM Employee
```

```
SELECT Last_Name, Salary_Amount, AvgSal
FROM Employee e, DeptSal d
WHERE e.Salary_Amount > d.AvgSal;
```

```
SELECT Last_Name, Salary_Amount, AvgSal
FROM Employee e CROSS JOIN DeptSal d
WHERE e.Salary_Amount > d.AvgSal;
```

This slide discusses using permanent tables in an interim or temporary fashion. In the example we create an actual table to store information (i.e., the average salary for all employees) into a real table. We then write a cross join (no equality join condition) that qualifies each employee's salary where it is greater than what is stored into the new table (a single row). Since the cross join compares each and every row to a single value (the average salary for all employees), the cross join become trivial in that a single row comparison is done for each employee.

One of the nice things about this strategy is that you can actually display the average salary, which can't be accomplished via a subquery. The bad thing about this strategy is that you need to create the table and then drop it after using it. This requires more steps, and it requires DDL, which is not terrific.

## “Derived” Tables

### Derived tables -

- are “database created” tables that are for a single query only.
- are discarded by the database when they are no longer required.
- are materialized into spool life.
- are referenced and treated as any permanent table by the database.
- must be defined by the author of the query, and require -
  - ✓ a table name
  - ✓ columns and their names
  - ✓ a SELECT that is used to populate the table

```
SELECT Last_Name,  
       Salary_Amount,  
       AvgSal  
FROM   Employee e,  
       (SELECT AVG(Salary_Amount)  
        FROM Employee) AS AvgT (AvgSal)  
WHERE  e.Salary_Amount > AvgT.AvgSal;
```

Query to populate AvgT

Table Name – “AvgT”

Column Name(s) – “AvgSal”

“Derived” tables are those that are created automatically by the database for the life of the query. The database creates the table for us and then drops the table when it is no longer required. To do this we need to provide the database with information necessary to use it, namely: a table name, column names, and information on what to put in it. The example on this slide illustrates how to accomplish this. The table is then materialized into spool and dropped when no longer needed.

## Forms of Derived Tables

```
SELECT Last_Name,  
       Salary_Amount,  
       AvgSal  
FROM   Employee e,  
       (SELECT AVG(Salary_Amount)  
        FROM Employee) AS AvgT (AvgSal)  
WHERE  e.Salary_Amount > AvgT.AvgSal;
```

```
SELECT Last_Name,  
       Salary_Amount,  
       AvgSal  
FROM   Employee e,  
       (SELECT AVG(Salary_Amount) AS AvgSal  
        FROM Employee) AS AvgT  
WHERE  e.Salary_Amount > AvgT.AvgSal;
```

```
SELECT Last_Name,  
       Salary_Amount,  
       AvgSal  
FROM   Employee e JOIN  
       (SELECT AVG(Salary_Amount) AS AvgSal  
        FROM Employee) AvgT  
ON     e.Salary_Amount > AvgT.AvgSal;
```

Each of these queries  
return the same result.

They each involve a  
different form of derived  
table usage.

There are various forms (or styles) one may use when using derived tables. This slide illustrates the most common forms. Many variations are possible, all of which are variations of those shown.

## Complex Derived Table Join

*Show the department name for those having a salary larger than their department average.*

```
SELECT  d.Department_Name, e.Last_Name, e.Salary_Amount, AvgT.AvgSal
FROM    Employee e,
        Department d,
        (SELECT Department_Number, AVG(Salary_Amount) AS AvgSal
         FROM Employee
         GROUP BY 1) AS AvgT
WHERE   e.Department_Number = d.Department_Number
AND     e.Department_Number = AvgT.Department_Number
AND     e.Salary_Amount > AvgT.AvgSal
ORDER BY 1;
```

department_name	last_name	salary_amount	AvgSal
customer support	Brown	43100.00	35545.83
customer support	Trader	37850.00	35545.83
customer support	Rogers	46000.00	35545.83
customer support	Johnson	36300.00	35545.83
education	Villegas	49700.00	38700.00
education	Brown	43700.00	38700.00
marketing sales	Wilson	53625.00	50031.25
marketing sales	Ratzlaff	54000.00	50031.25
marketing sales	Runyon	66000.00	50031.25
research and development	Stein	29450.00	29350.00

The query in this slide simply adds another table to the mix. By adding the department name, we must involve the department table. So there are now three tables being joined together:

- Employee (for getting last name, first name, and salary amount)
- Department (for getting department name)
- AvgT (for deriving the average salary for each department)

Of course, there are all of the necessary one-to-many joins needed for correctly obtaining the result.

- Department number is unique in department (one-to-many to employee)
- Department number is unique in AvgT (one-to-many to employee)

## “WITH” Form of a Derived Table

More common usage?

```
SELECT Last_Name,
       Salary_Amount,
       AvgSal
FROM   Employee e,
       (SELECT AVG(Salary_Amount)
        FROM Employee) AS AvgT (AvgSal)
WHERE  e.Salary_Amount > AvgT.AvgSal;
```

Query to populate AvgT

Table Name – “AvgT”

Column Name(s) – “AvgSal”

Column Name(s) – “AvgSal”

WITH Form

Table Name – “AvgT”

Query to populate AvgT

The SELECT (*projection*) appears after the derived table definition.

```
WITH AvgT (AvgSal) AS
  (SELECT AVG(Salary_Amount)
   FROM Employee)
SELECT Last_Name,
       Salary_Amount,
       AvgSal
FROM   AvgT t, Employee e
WHERE  e.Salary_Amount > t.AvgSal;
```

Now we introduce the second form while contrasting it with the previous form. This second form of a derived table is often referred to as the “WITH” form. This form has a variation on it that will become very important in a later module that discusses “Recursive Queries,” which uses a recursive structure for the WITH form. Recursive queries cannot be performed by using the (more common) “FROM” form.

Notice that the WITH form is structured completely “upside-down” from its counterpart.

“WITH” form:

- Definition appears at the top of the query, prior to the SELECT portion.

“FROM” form:

- Definition appears in the FROM portion, after the SELECT portion.

## Multiple “WITH” Derived Tables

The following is an example of how to use multiple WITH derived tables in the same query.

```
WITH AvgS (AvgSal) AS  
(SELECT AVG(Salary_Amount)  
FROM Employee),  
AvgB (AvgBudget) AS  
(SELECT SUM(Budget_Amount)  
FROM Department)  
SELECT AvgSal, AvgBudget  
FROM AvgS , AvgB;
```

One WITH statement.

Each derived table definition is separated by a comma.

Lastly, the query that references them.

## Including the NULL Department

```
SELECT e.Department_Number,
       e.Last_Name,
       e.Salary_Amount,
       AvgT.AvgSal
FROM   Employee e,
       (SELECT Department_Number, AVG(Salary_Amount) AS AvgSal
        FROM Employee
        GROUP BY 1) AS AvgT
WHERE  COALESCE(e.Department_Number, -1) = COALESCE(AvgT.Department_Number, -1)
AND    e.Salary_Amount > AvgT.AvgSal
ORDER BY 1;
```

department_number	last_name	salary_amount	AvgSal
-----	-----	-----	-----
?	Rogers	56500.00	43316.67
301	Stein	29450.00	29350.00
401	Trader	37850.00	35545.83
401	Brown	43100.00	35545.83
401	Johnson	36300.00	35545.83
401	Rogers	46000.00	35545.83
403	villegas	49700.00	38700.00
403	Brown	43700.00	38700.00
501	wilson	53625.00	50031.25
501	Runyon	66000.00	50031.25
501	Ratzlaff	54000.00	50031.25

An outer join is not comparable because, in an employee-to-employee join, all departments should match except null.

For this slide, here is what an outer join would return. Basically the result is that of only the inner join (without “Rogers”) who is the only employee with a null department and a salary greater than the average salary for the null department.

```
SELECT e.Department_Number,
       e.Last_Name,
       e.Salary_Amount,
       AvgT.AvgSal
FROM   Employee e LEFT JOIN
       (SELECT Department_Number, AVG(Salary_Amount) AS
       AvgSal
        FROM Employee
        GROUP BY 1) AS AvgT
ON     e.Department_Number = AvgT.Department_Number
WHERE  e.Salary_Amount > AvgT.AvgSal
ORDER BY 1;
```

department_number	last_name	salary_amount	AvgSal
-----	-----	-----	-----
301	Stein	29450.00	29350.00
401	Brown	43100.00	35545.83

35545.83	401	Johnson	36300.00
35545.83	401	Trader	37850.00
35545.83	401	Rogers	46000.00
38700.00	403	Villegas	49700.00
38700.00	403	Brown	43700.00
50031.25	501	Wilson	53625.00
50031.25	501	Runyon	66000.00
50031.25	501	Ratzlaff	54000.00

## Module 16: Summary

- Permanent tables use permanent space and must be physically created and dropped.
- Derived tables are created and dropped by the database.
- Table level attributes have default values.
- Some column level attributes are defaulted while most must be specified.
- Secondary indexes may be included into the CREATE TABLE or separately after the table has been created and/or loaded.
- Derived tables are specified in the FROM clause of a SQL request.
- Derived tables must specify a table name, a column name list, and tell the database what to load into it.



## Module 25: Derived Tables and Volatile Tables

Teradata SQL for Business Users

Copyright © 2007–2020 by Teradata. All Rights Reserved.

## Derived Tables and Volatile Tables

After completing this module, you will be able to:

- Recognize variations for both forms of Derived table syntax.
- Create volatile tables for session use.
- Distinguish between various ad-hoc strategies.
- Identify volatile table limitations.
- Distinguish between the two ON COMMIT options.
- Use volatile tables within views and macros.

## Temporary Table Choices

### Views

- Local to a query
- Uses Spool
- May be replaced with derived tables

### Derived Tables

- Local to the query
- Incorporated into SQL query syntax
- Discarded when query finishes
- No Data Dictionary involvement
- May be replaced with views

### Volatile Tables

- Local to a session (*are available to all queries during the session*)
- Uses CREATE VOLATILE TABLE syntax
- Discarded automatically at session end
- No Data Dictionary involvement

### Global Temporary Tables (*not taught in this course*)

- Local to a session (*like volatile tables*)
- Uses CREATE GLOBAL TEMPORARY TABLE syntax (*e.g., a DBA creates the definition*)
- Materialized instance of table discarded at session end (*like volatile tables*)
- Creates and keeps table definition in Data Dictionary (*i.e., the DBA created table*)

In comparing the choice in this slide, one should be reminded that spool files cannot survive a restart. This means that upon a database restart, the volatile tables goes away, so you will need to create the table again and repeat the process that lead to the point of failure. This implies an ad-hoc usage, where a person is visually active with the process while it is processing.

### Views

The definition of a View exists after the query is finished but it is “materialized” in spool each time the query is executed. When the query has completed, this materialized version of the View is removed when the spool file is dropped.

### Global Temporary Tables

Global temporary table usage is such that “Temp” space can survive a restart, and can be used in a scripted application that is run unattended on a scheduled basis. Global Temporary tables are significantly more complex than are Volatile temp tables, and are typically considered more a concern of application development than a SQL concern.

### Derived Tables

Derived tables also use spool, and do not survive a database restart, nor do they survive a request failure. Volatile and Global temp tables can be created to survive request failures but do not by default.

## Another Derived Table Syntax Form

More common usage?

```
SELECT Last_Name,
       Salary_Amount,
       AvgSal
FROM   Employee e,
       (SELECT AVG(Salary_Amount)
        FROM Employee) AS AvgT (AvgSal)
WHERE  e.Salary_Amount > AvgT.AvgSal;
```

Query to populate AvgT

Table Name – “AvgT”

Column Name(s) – “AvgSal”

Column Name(s) – “AvgSal”

WITH Form

Table Name – “AvgT”

Query to populate AvgT

The SELECT (*projection*) appears after the derived table definition.

```
WITH AvgT (AvgSal) AS
  (SELECT AVG(Salary_Amount)
   FROM Employee)
SELECT Last_Name,
       Salary_Amount,
       AvgSal
FROM   AvgT t, Employee e
WHERE  e.Salary_Amount > t.AvgSal;
```

Now we introduce the second form while contrasting it with the previous form. This second form of derived table is often referred to as the “WITH” form. This form has a variation on it that will become very important in a later module that discusses “Recursive Queries,” which uses a recursive structure for the WITH form. Recursive queries cannot be performed by using the (more common?) “FROM” form.

Notice that the WITH form is structured completely “upside-down” from its counterpart.

“WITH” form:

- Definition appears at the top of the query, prior to the SELECT portion.

“FROM” form:

- Definition appears in the FROM portion, after the SELECT portion.

## Volatile Table Syntax

```
CREATE VOLATILE TABLE vt_deptsal
(deptno  SMALLINT
,avgsal  DEC(9,2)
,maxsal  DEC(9,2)
,minsal  DEC(9,2)
,sumsal  DEC(9,2)
,empcnt  SMALLINT);
```

Volatile tables are not defined in ANSI

```
SHOW TABLE vt_deptsal;
```

```
CREATE SET VOLATILE TABLE XYZ.vt_deptsal ,
FALLBACK ,
CHECKSUM = DEFAULT,
LOG
(
deptno SMALLINT,
avgsal DECIMAL(9,2),
maxsal DECIMAL(9,2),
minsal DECIMAL(9,2),
sumsal DECIMAL(9,2),
empcnt SMALLINT)
PRIMARY INDEX ( deptno )
ON COMMIT DELETE ROWS;
```

- [LOG](#) indicates that a transaction journal is maintained. This is the default.
- [NO LOG](#) allows for better performance.
- [ON COMMIT DELETE ROWS](#) indicates to delete all table rows after a commit (*end transaction*). This is the default.
- [ON COMMIT PRESERVE ROWS](#) indicates to keep table rows at TXN end.

Volatile tables do not have a persistent definition; they must be newly created each time you need to use them. The table definition is cached only for the duration of the session in which it is created.

If you frequently reuse particular volatile table definitions, consider writing a macro that contains the CREATE TABLE text for those volatile tables. Because volatile tables are private to the session that creates them, the system does not check their creation, access, modification, and drop privileges. Any user that has spool can create them. Understand that by holding on to the spool for the life of the session, the user has less spool available to them for other queries.

The following list details the general characteristics of volatile tables:

- Both the contents and the definition of a volatile table are destroyed when a system reset occurs.
- Space usage is charged to the login user spool space.
- A single session can materialize up to 1,000 volatile tables at one time.
- The primary index for a volatile table can be either an NPPI or a PPI.
- You cannot create secondary, hash, or join indexes on a volatile table.
- You cannot collect statistics on volatile table columns, including the PARTITION column of a PPI volatile table.

## Volatile Table Restrictions

- Up to 1000 volatile tables are allowed for a single session.
- At the time you create a volatile table, the name must be unique among all global and permanent object names in the database that has the name of the login user.

```
CREATE VOLATILE TABLE username.table1
CREATE VOLATILE TABLE table1
CREATE VOLATILE TABLE databasename.table1
```

Explicit username must be that of logon.

Default database is ignored.

Explicit database name cannot use spool of another database or user.

Each session can use the same VT name (local to session).

VT name cannot duplicate existing object name for this user

- Perm or Temp table names
- View names
- Macro names
- Trigger names, etc.

The following options are not permitted for volatile tables:

- Referential integrity constraints
- CHECK constraints
- Permanent journaling
- Compressed column values
- DEFAULT clause
- TITLE clause
- Named indexes

Volatile tables always use spool directly from creating a user's spool definition. It is for this reason that, if you specify another database name, it will fail. If your default database is set to some database other than your user name, and you don't qualify the database name in the create (taking your default), the creation of the volatile table is successful because the default is ignored.

## HELP and SHOW (Volatile) TABLE

```
CREATE VOLATILE TABLE vt_deptsal1
(
  deptno SMALLINT,
  avgsal DECIMAL(9,2),
  maxsal DECIMAL(9,2),
  minsal DECIMAL(9,2),
  sumsal DECIMAL(9,2),
  empcnt SMALLINT );
```

HELP DATABASE command does not show VT's and they do not appear in the Explorer Tree window in SQL Assistant.

```
SHOW TABLE vt_deptsal1;
```

```
HELP VOLATILE TABLE;
```

Table Name	Table Id
vt_deptsal1	30C0BC140000
vt_deptsal2	30C0BD140000

```
DROP TABLE vt_deptsal1;
```

```
CREATE SET VOLATILE TABLE XYZ.vt_deptsal1 ,
  FALLBACK ,
  CHECKSUM = DEFAULT,
  LOG
(
  deptno SMALLINT,
  avgsal DECIMAL(9,2),
  maxsal DECIMAL(9,2),
  minsal DECIMAL(9,2),
  sumsal DECIMAL(9,2),
  empcnt SMALLINT)
PRIMARY INDEX ( deptno )
ON COMMIT DELETE ROWS;
```

When creating a volatile table, the default is ON COMMIT DELETE ROWS. This means that when a transaction ends, the rows are automatically deleted from the table. The same volatile table in this slide would be like creating it with this syntax.

```
CREATE VOLATILE TABLE
(deptno      SMALLINT,
Avgsal      DECIMAL(9,2),
maxsal      DECIMAL(9,2),
minsal      DECIMAL(9,2),
sumsal      DECIMAL(9,2),
empcnt      SMALLINT)
PRIMARY INDEX (deptno)
ON COMMIT DELETE ROWS;
```

The default would also include that it is a SET table with LOG on. A defaulted primary index would also result in the first column of the table being a NUPI (non-unique primary index).

## ON COMMIT DELETE ROWS (Implicit Transactions)

teradata.

Create a volatile table.

```
CREATE VOLATILE TABLE vt_deptsal
(deptno SMALLINT
,avgsal DEC(9,2)
,maxsal DEC(9,2)
,minsal DEC(9,2)
,sumsal DEC(9,2)
,empcnt SMALLINT);
```

Populate the volatile table with computed aggregates.

```
INSERT INTO vt_deptsal
SELECT dept ,AVG(sal) ,MAX(sal) ,MIN(sal), SUM(sal), COUNT(emp)
FROM emp
GROUP BY 1;
```

```
SELECT * FROM vt_deptsal
ORDER BY 3;
```

\*\*\* Query completed. No rows found.

Remember: The default is ON COMMIT DELETE ROWS.  
Rows are deleted immediately after the insert for implicit transactions!

For Teradata mode, transaction processing is implicit. This means that each request is automatically a commit point (or an “implied” transaction).

Since the default is ON COMMIT DELETE ROWS, and the request is an implied commit, the moment the rows are inserted they are deleted due to the commit.

## ON COMMIT PRESERVE ROWS (Implicit Transactions)

teradata.

1) Create a volatile table.

```
CREATE VOLATILE TABLE vt_deptsal
(deptno SMALLINT
,avgsal DEC(9,2)
,maxsal DEC(9,2)
,minsal DEC(9,2)
,sumsal DEC(9,2)
,empcnt SMALLINT)
ON COMMIT PRESERVE ROWS;
```

2) Populate the volatile table with computed aggregates.

```
INSERT INTO vt_deptsal
SELECT dept ,
        AVG(sal) ,
        MAX(sal) ,
        MIN(sal),
        SUM(sal),
        COUNT(emp)
FROM emp_views.emp
GROUP BY 1;
```

3) `SELECT * FROM vt_deptsal ORDER BY 3;`

deptno	avgsal	maxsal	minsal	sumsal	empcnt
301	29350.00	29450.00	29250.00	58700.00	3
401	35545.83	46000.00	24500.00	213275.00	7
403	38700.00	49700.00	31000.00	193500.00	6
402	52500.00	52500.00	52500.00	52500.00	2
?	43316.67	56500.00	34700.00	129950.00	3
501	50031.25	66000.00	26500.00	200125.00	4
999	100000.00	100000.00	100000.00	100000.00	1

Alternately, you can choose to commit your inserts, updates, and deletes by specifying so in your create volatile table syntax as shown in this slide.

## ON COMMIT DELETE ROWS (Explicit Transactions)

teradata.

Create a volatile table.

```
CREATE VOLATILE TABLE vt_deptsal
(deptno SMALLINT
,avgsal DEC(9,2)
,maxsal DEC(9,2)
,minsal DEC(9,2)
,sumsal DEC(9,2)
,empcnt SMALLINT);
```

```
BT;
INSERT INTO vt_deptsal (1, 2, 3, 4, 5, 6);
SELECT * FROM vt_deptsal;
```

deptno	avgsal	maxsal	minsal	sumsal	empcnt
1	2.00	3.00	4.00	5.00	6

```
ET;
SELECT * FROM vt_deptsal;

*** Query completed. No rows found.
```

The default of ON COMMIT DELETE ROWS deleted the rows immediately after the ET;

This would work the same for ANSI mode explicit transactions by using "COMMIT WORK."

For “explicit” transactions, we tell the database when to perform the commit by issuing a BEGIN TRANSACTION and following that with an END TRANSACTION. These two statements bound a number of requests making them commit when we decide – explicitly!

Since the commit is withheld until an ET statement is encountered, the rows will be deleted on our terms, and then the table will become empty.

## ON COMMIT PRESERVE ROWS (Explicit Transactions)

teradata.

Create a volatile table.

```
CREATE VOLATILE TABLE vt_deptsal  
(deptno SMALLINT  
,avgsal DEC(9,2)  
,maxsal DEC(9,2)  
,minsal DEC(9,2)  
,sumsal DEC(9,2)  
,empcnt SMALLINT)  
ON COMMIT PRESERVE ROWS;
```

```
BT;
```

```
INSERT INTO vt_deptsal (1, 2, 3, 4, 5, 6);
```

```
SELECT * FROM vt_deptsal;
```

deptno	avgsal	maxsal	minsal	sumsal	empcnt
1	2.00	3.00	4.00	5.00	6

```
ET;
```

```
SELECT * FROM vt_deptsal;
```

deptno	avgsal	maxsal	minsal	sumsal	empcnt
1	2.00	3.00	4.00	5.00	6

We may also elect to have the database never delete the rows of the volatile table until we issue a DELETE DML statement. With this approach the rows of the table will never be “automatically” deleted on a commit, but instead the rows will be deleted under our control via a DELETE command.

## Limitations

The following commands are not applicable to VT's:

- CREATE/DROP INDEX
- ALTER TABLE
- GRANT/REVOKE privileges
- DELETE DATABASE/USER (does not drop VT's)

*VT's may not:*

- Use ACCESS LOGGING
- Be RENAMEd
- Be loaded with MultiLoad or FastLoad utilities

Along with the limitations in this slide, recall the restrictions from an earlier slide.

*The following options are not permitted for volatile tables:*

- *Referential integrity constraints*
- *CHECK constraints*
- *Permanent journaling*
- *Compressed column values*
- *DEFAULT clause*
- *TITLE clause*
- *Named indexes*

## Using INSERT-SELECT

You can populate an empty table using INSERT-SELECT.

There are two basic forms of INSERT-SELECT.

1) `INSERT INTO targettable SELECT * FROM sourcetable;`

The SELECT may include derived data values or involve any number of features and functions as expressions.

2) `INSERT INTO targettable  
SELECT column1, column2, , , ,  
FROM sourcetable;`

Things to consider are:

- Match the number of source and target columns.
- Match the data types or implicit conversions may occur.

The target table may be populated or empty.  
Empty target tables populate faster than populated ones.

## Inserting a Single Row

Insert a new employee into the employee table.

Value order assumes table column order.

```
INSERT INTO employee  
VALUES      (1210, NULL, 401, 412101, 'Smith', 'James', 890303, 460421, 41000);
```

Insert a new employee with only partial data.

Column list may be any order.

Value list must match order of column list.

Inserts default values (discussed later) for missing columns.

```
INSERT INTO employee  
  (last_name, first_name, hire_date, birthdate, salary_amount, employee_number)  
VALUES      ('Garcia', 'Maria', 861027, 541110, 76500.00, 1291);
```

As a Teradata Extension to ANSI:

- INSERT can be abbreviated as INS.
- INTO and VALUES are optional keywords.

There is no typing shortcut facility for explicitly inserting multiple rows with this form.

This slide discusses the ability of SQL to insert a single row into a table. The two forms provide alternate methods for performing an insert. The first method is used if you know the order of the columns in the table create statement, and assumes that you know their data types as well. The second form also assumes that you know the data types for each column, but, rather than knowing the table's column order, you reference the columns by name. This means that you do not need the order because the database can now match the columns by name. However you must list the order of their values so that they match the order in the column list.

## UPDATE

UPDATE modifies one or more rows in a single table.

EMPLOYEE								
EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1003	401	412101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

For employee 1004, change her:

- Department to 403
- Job to 432101
- Manager to 1005

```
UPDATE Employee [ FROM Employee ]
SET    Department_Number = 403
      ,Job_Code           = 432101
      ,Manager_Employee_Number = 1005
WHERE  Employee_Number   = 1004;
```

EMPLOYEE								
EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1005	403	432101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

The UPDATE clause is used to change data values for columns. The example shown is simple enough since the predicate is specifying that we are updating only a single row (referencing a UPI value like Employee\_Number guarantees this).

Updating may also be done in bulk. In this case “bulk” means more than one row. An example of a bulk operation would be if the WHERE condition was to be removed from this update so that all rows would be updated. Another example of a bulk operation is provided in the next discussion on the following slide.

The FROM clause is optional. Optional keywords are often referred to as being “noise” (something that can be disregarded or ignored).

Examples of using the DEFAULT keyword in an update follow.

```
UPDATE
EMPLOYEE SET Last_Name = DEFAULT
WHERE Salary_Amount = DEFAULT;
```

```
*** Failure 3811 Column 'last_name' is NOT NULL. Give the column a value.
```

```
UPDATE  
EMPLOYEE SET Department_Number = DEFAULT  
WHERE Salary_Amount = DEFAULT;
```

## Updating with Joins

Updates with joins and subqueries allow a table's rows to be updated based on information in another table.

*Give everyone in all the support departments a 10% raise.*

*(Assume we don't know the department numbers for all of the support departments.)*

Using a subquery:

```
UPDATE employee
SET salary_amount = salary_amount * 1.10
WHERE department_number IN
  (SELECT department_number
   FROM department
   WHERE department_name LIKE '%Support%');
```

Using an inner join:

```
UPDATE employee [ FROM Department ]
SET salary_amount = salary_amount * 1.10
WHERE employee.department_number =
      department.department_number
AND department_name LIKE '%Support%';
```

Using a correlated subquery:

```
UPDATE employee e
SET salary_amount = salary_amount * 1.10
WHERE department_number =
  (SELECT department_number
   FROM department d
   WHERE e.department_number =
         d.department_number
   AND department_name LIKE '%Support%');
```

Updating by using joins (subqueries are joins) is considered a bulk operation event, though it may result in only affecting a single row since the process for updating that row may not necessarily be considered a direct one.

## DELETE

DELETE removes one or more rows from a table.



EMPLOYEE								
EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1003	401	412101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

Remove employees in department 301 from the employee table.

```
DELETE FROM employee
WHERE department_number = 301;
```

All of these are equivalent and empty a table.

```
DELETE FROM emp_data ALL;
DELETE FROM emp_data;
DELETE emp_data;
DEL emp_data;
```

The DELETE clause can be used to remove rows from a table. When used conditionally, as in the first example, it can be used to remove targeted rows based upon the explicit values referenced in the predicate. The examples at the bottom are repeated from an earlier module, and remove all rows from the table very quickly. The keyword “FROM” is noise, and is not required, and the keyword DELETE may be abbreviated to “DEL.”

You may also use the DEFAULT keyword in a delete like this.

```
DELETE FROM EMPLOYEE WHERE Salary_Amount = DEFAULT;
```

## Deleting with Joins

*Remove all of the employees who are assigned to a temporary department.  
(i.e., for which the department name is "Temp.")*

Using a  
Subquery:

```
DELETE FROM employee
WHERE      department_number IN
          (SELECT    department_number
           FROM      department
           WHERE      department_name = 'Temp');
```

Using a  
Join:

```
DELETE FROM employee
WHERE      employee.department_number =
          department.department_number
AND        department.department_name = 'Temp';
```

Using a  
Correlated  
Subquery:

```
DELETE FROM employee e
WHERE      department_number =
          (SELECT department_number
           FROM  department d
           WHERE e.department_number = d.department_number
           AND   d.department_name = 'Temp');
```

As seen earlier with UPDATE, joins may be used to delete rows from a table. The keyword "FROM" is noise, and is not required, and the keyword DELETE may be abbreviated to "DEL."

## Module 25: Summary

- There are two forms for writing derived tables; WITH and FROM.
- Views, Derived, Global, and Volatile tables are all examples of temporary instance objects.
- Views, Derived tables, and Volatile tables all use spool.
- ON COMMIT DELETE ROWS is the default for volatile tables.
- Volatile tables last until the end of the user's session logoff, when the database drops them automatically.
- Volatile tables are best suited for ad-hoc usage.

# 2.0 Data Visualization & Presentation

# Moving Data from Teradata Database to an External Target

**Source:** <https://docs.teradata.com/reader/j9~8T4F8ZcLkW7Ke0mxgZQ/VTx~L6axIqWGM02RInPtzg>

This chapter describes several methods for using Teradata PT to move data from a Teradata Database into a non-Teradata target. It includes the following topics:

▢ [Data Flow Description](#)

▢ [Comparing Applicable Operators](#)

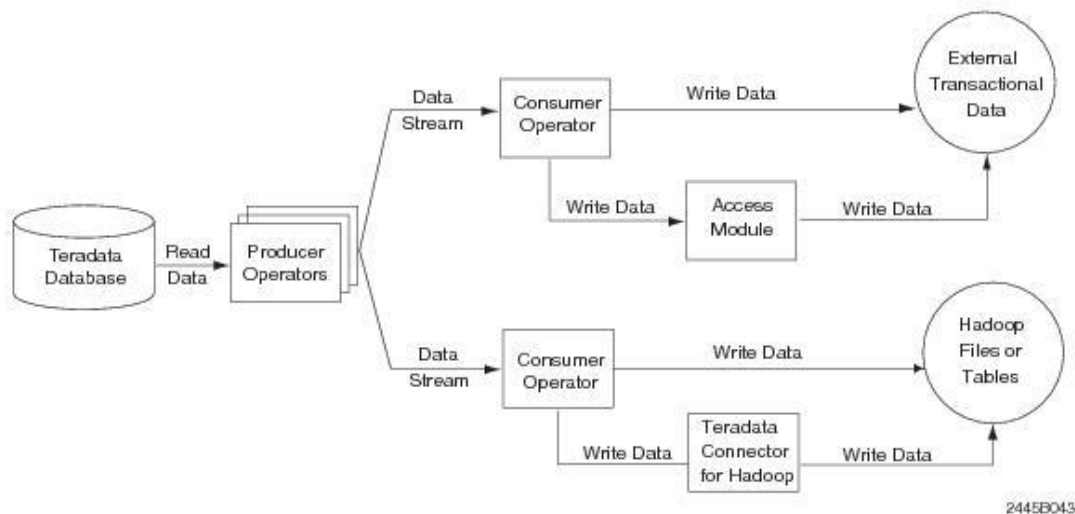
▢ [Using Access Modules to Process Data Before Writing to External Targets](#)

▢ [Common Data Movement Jobs](#)

## Data Flow Description

Teradata PT offers several paths for moving data from a Teradata Database into a non-Teradata target, as shown in the following composite diagram.

Figure 36: Moving Data from a Teradata Database into a Non-Teradata Target



Note that many of the blocks in [Figure 36](#) allows you to choose among several operators and access modules. Read the following sections to understand how to make the best choices for specific data movement jobs.

## Comparing Applicable Operators

Once you identify the requirements for moving data from Teradata Database to an external data source, you must select the components that the script will use to execute the job. There are three types of components you need to consider:

▢ A producer operator that reads data from a Teradata Database and places it in the data stream.

and

❑ A consumer operator that takes data from the data stream and writes it to the data target.

or

❑ A consumer operator that uses an OUTMOD routine or access module to post-process the data before loading the data target.

### Producer Operators

The Teradata PT producer operators in this section read data from a Teradata Database and write it to the data stream.

The Teradata PT job script invokes a producer operator, which employs the user-specified SQL SELECT statement to access Teradata Database tables. For further information on using APPLY/SELECT to specify a producer operator, see [“Coding the APPLY Statement” on page 64](#) and the section on APPLY in *Teradata Parallel Transporter Reference*.

The following table briefly describes and compares the function of each Teradata PT operator that can be used as a producer when extracting data from a Teradata Database:

Operator	Description
Export Operator	<p>Extracts large volumes of data from a Teradata Database at high speed. Function is similar to the standalone Teradata FastExport utility.</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>❑ Allows use of multiple parallel instances.</li> <li>❑ For a sorted answer set, redistribution of the rows occurs over the BYNET. This allows for easy recombination of the rows and data blocks when they are sent to the client in sorted order.</li> </ul> <p><b>Limitations:</b></p> <ul style="list-style-type: none"> <li>❑ Cannot be used to retrieve data in TEXT mode and write it to target files in the TEXT or VARTEXT (delimited) format. Use SQL Selector for this where possible.</li> <li>❑ A sorted answer set requires that only a single instance of the Export operator can be used. Specifying ORDER BY in the SELECT statement and multiple Export operator instances results in an error.</li> </ul>

	For details, see <i>Teradata Parallel Transporter Reference</i> .
SQL Selector Operator	<p>Submits a single SQL SELECT statement to the Teradata Database to retrieve data from a table.</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>☐ Use to retrieve data in TEXT mode and write it to target files in the TEXT or VARTEXT (delimited) format.</li> <li>☐ Can retrieve LOB, JSON and XML data from the Teradata Database.</li> </ul> <p><b>Limitations:</b></p> <ul style="list-style-type: none"> <li>☐ Much slower than Export operator.</li> </ul> <p>Teradata strongly recommends that you specify XMLSERIALIZE on selected XML columns so that the byte-order-mark (BOM) matches the XML encoding when using the client UTF-16 character set.</p> <p>For details, see <i>Teradata Parallel Transporter Reference</i>.</p>

## Consumer Operators

The Teradata PT consumer operators in this section read data from the data stream and write it to an external target.

The Teradata PT job script invokes a consumer operator using an APPLY statement. For further information on using SELECT to specify a producer operator, see [“Coding the APPLY Statement” on page 64](#) and the section on APPLY in *Teradata Parallel Transporter Reference*.

The following table briefly describes and compares the function of each Teradata PT operator that can be used as a consumer when moving data from Teradata Database to an external data target:

Operator	Description
<b>Operators that Write Data to a non-Teradata Target</b>	
DataConnector Operator	<p>Writes data to flat files and functions similarly to the DataConnector standalone utility.</p> <p><b>Features:</b></p>

	<p>❓ Can write directly to an external file or through an access module.</p> <p>❓ Writes to files and tables in Hadoop.</p> <p><b>Limitations:</b></p> <p>❓ Cannot write ZIP and GZIP files to a Hadoop/HDFS data source.</p> <p>For details, see <i>Teradata Parallel Transporter Reference</i>.</p>
<b>Operators that Pre-process Data before Writing to a non-Teradata Target</b>	
FastExport OUTMOD Adapter Operator	<p>Uses a FastExport OUTMOD routine to pre-process data before writing it to the data target.</p> <p>For details, see <i>Teradata Parallel Transporter Reference</i>.</p>

#### Using Access Modules to Process Data Before Writing to External Targets

Access modules are dynamically attached software components of the Teradata standalone load and unload utilities. Some access modules are usable with Teradata PT job scripts, and provide the input/output interface between operators and various types of external data storage devices. Any operator that uses access modules can interface with *all* available access modules.

The following access modules can be used as part of a job to move data from Teradata Database to an external data target.

Access Module	Description
OLE DB	Provides write access to a flat file or a table in an OLE DB-compliant DBMS, such as SQL Server, Oracle or Connix.

#### Specifying an Access Module

Use the AccessModuleName attribute in the DataConnector (consumer) operator to specify the optional use of an access module to interface with the target database. The DataConnector operator definition must also specify a value for the AccessModuleInitStr attribute, to define the access module initialization string.

For detailed information on requirements for using access modules with Teradata PT, see *Teradata Tools and Utilities Access Module Reference*.

For information on using access modules with z/OS, see [“Using Access Modules to Read Data from an External Data Source” on page 98](#).

## Using the DataConnector Operator to Write Files and Tables in Hadoop

In addition to writing flat files and interfacing with access modules, the DataConnector operator also has the ability to write to Hadoop files and tables. The following table briefly describes and compares the two interfaces which the DataConnector operator can use to move data from the data stream to Hadoop files and tables.

Interface	Description
HDFS API	Provides access to Hadoop files via the Hadoop Distributed File System Application Programming Interface, or HDFS API. The HDFS is a POSIX-compatible file system with some minor restrictions. It does not support updating files and it only supports writing files in truncate mode or append mode. The Hadoop Software is written in Java and the HDFS API is a Java JNI interface that exposes all the expected standard posix file system interfaces for reading and writing HDFS files directly by a C/C++ program. The Data Connector Producer and Consumer operators have been updated to directly access the HDFS file system using the HDFS API. All standard Data Connector file system features are supported.
"TDCH-TPT	Provides access to Hadoop files and tables via the Teradata Connector for Hadoop, or TDCH. TDCH utilizes the MapReduce framework's distributed nature to transfer large amounts of data in parallel from Hadoop files and tables to the DataConnector operator. The TDCH-TPT interface gives TPT users the ability to read and write HDFS files, Hive tables, and Hcat tables in various Hadoop-specific formats. Because this interface relies on TDCH to read and write data, many of the traditional DataConnector attributes are unsupported.

For information, see the section "Processing Hadoop Files and Tables" in Teradata Parallel Transporter Reference.

**Note:** GZIP and ZIP files are not supported with Hadoop/HDFS.

**Note:** HDFS processing can be activated simply by adding the following attribute to a Data Connector Consumer or Producer:

HadoopHost = 'default'

### Common Data Movement Jobs

You can use any valid combination of producer and consumer operators, and where necessary access modules, to create a job script for your data movement needs. However, the following list includes examples of some of the most common job scenarios. Evaluate the examples and if possible use one of the associated sample job scripts before creating your own.

[!\[\]\(3570ab8e0b6f87d2cdce366e6ea5559f\_img.jpg\) Job Example 12: Extracting Rows and Sending Them in Delimited Format](#)

❏ [Job Example 13: Extracting Rows and Sending Them in Indicator-mode Format](#)

❏ [Job Example 14: Export Data and Process It with an OUTMOD Routine](#)

❏ [Job Example 15: Export Data and Process It with an Access Module](#)

❏ [Job Example 16: Extract BLOB/CLOB/JSON Data and Write It to an External File](#)

❏ [Job Example 17: Extract Rows and Write Them to a Hadoop File](#)

❏ [Job Example 18: Extract Rows and Write Them to a Hadoop Table](#)

## Job Example 12: Extracting Rows and Sending Them in Delimited Format

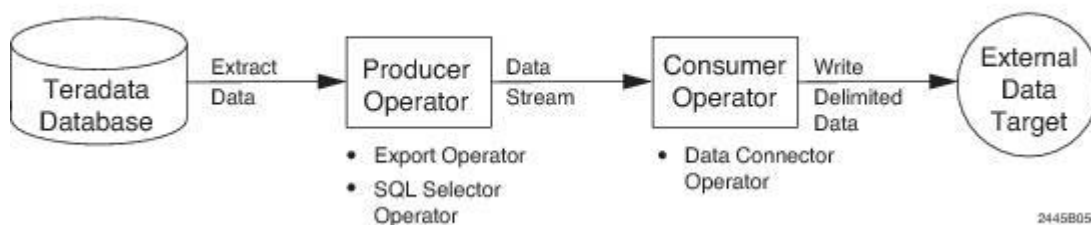
### Job Objective

Extract rows from Teradata Database tables and write them to an external target file as delimited data.

### Data Flow Diagram

Figure 37 shows a diagram of the job elements for Job Example 10.

Figure 37: Job Example PTS00016, PTS00017 -- Extracting Rows and Sending Them in Delimited Format



### Sample Script

For the sample script that corresponds to this job, see the following scripts in the sample/userguide directory:

**PTS00016:** Extracting Rows and Writing Them in Delimited Format using the Export operator.

**PTS00017:** Extracting Rows and Writing Them in Delimited Format using the SQL Selector operator.

### Rationale

This job uses the:

- ❏ Export operator for exporting data from a Teradata Database table with the schema that matches the table.
- ❏ SQL Selector operator for extracting data from a Teradata Database table in field mode (character format).

❑ DataConnector operator because it is the only operator that can write character data to an external flat file in delimited format.

### Job Example 13: Extracting Rows and Sending Them in Indicator-mode Format

### Job Example 13: Extracting Rows and Sending Them in Indicator-mode Format

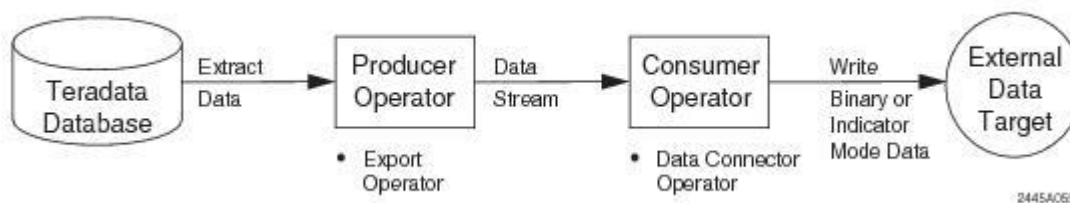
#### Job Objective

Extract rows from Teradata Database tables using Export operator and write them to an external target as indicator-mode data.

#### Data Flow Diagram

Figure 38 shows a diagram of the job elements for Job Example 11.

Figure 38: Job Example PTS00018 -- Extracting Rows and Sending Them in Binary or Indicator-mode Format



#### Sample Script

For the sample script that corresponds to this job, see the following scripts in the sample/userguide directory:

**PTS00018:** Exporting Rows and Writing Them as Binary or Indicator Mode Data.

#### Rationale

This job uses the operators shown for the following reasons:

❑ Use Export operator because it can extract large amounts of data from a Teradata Database table at high speeds.

❑ DataConnector operator because it can write data to an external flat file.

### Job Example 14: Export Data and Process It with an OUTMOD Routine

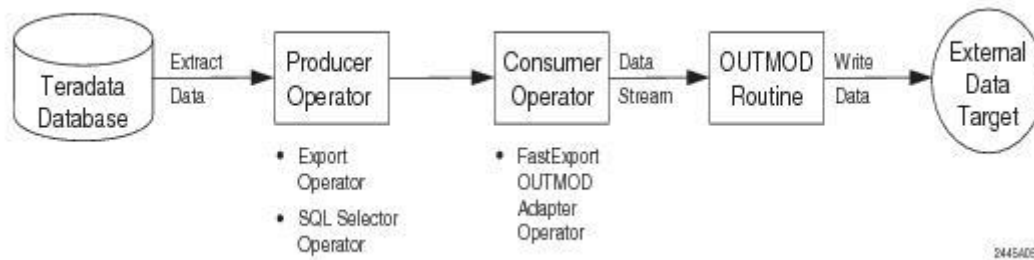
#### Job Objective

Export data from a Teradata Database table and send it to an OUTMOD for post-processing before loading into an external target. This job is applicable to OUTMODs written for the FastExport utility.

## Data Flow Diagram

Figure 39 shows a diagram of the job elements for Job Example 12.

Figure 39: Job Example PTS00019 -- Export Data and Process It with an OUTMOD Routine



## Sample Script

For the sample script that corresponds to this job, see the following scripts in the sample/userguide directory:

**PTS00019:** Exporting Data and Processing It with an OUTMOD Routine.

## Rationale

The job uses:

☑ Export operator because it is the fastest way to extract large amounts of data from a Teradata Database.

**Note:** The SQL operator extracts data more slowly than the Export operator. Use the SQL Selector operator only if the Teradata Database is short on load slots, because the SQL Selector operator does not use Teradata Database load slots.

☑ FastExport OUTMOD Adapter because it is the only operator that can interface with an OUTMOD routine written for the FastExport utility.

## Job Example 15: Export Data and Process It with an Access Module

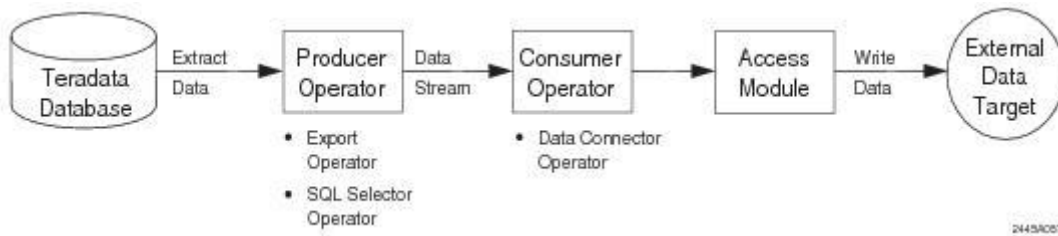
### Job Objective

Export rows from a Teradata Database table and send them to an Access Module for processing before loading the data into an external target.

## Data Flow Diagram

Figure 40 shows a diagram of the job elements for Job Example 13.

Figure 40: Job Example PTS00020 --- Export Data and Process It with an Access Module



## Sample Script

For the sample script that corresponds to this job, see the following script in the sample/userguide directory:

**PTS00020:** Exporting Data and Processing It with an Access Module.

## Rationale

The job uses:

❑ Export operator because it is the fastest at extracting large amounts of data from a Teradata Database.

**Note:** The SQL operator extracts data more slowly than the Export operator. Use the SQL Selector operator only if the Teradata Database is short on load slots, because the SQL Selector operator does not use Teradata Database load slots.

❑ DataConnector operator because it is the only consumer operator that can interface with all Teradata PT-supported access modules.

## Job Example 16: Extract BLOB/CLOB/JSON Data and Write It to an External File

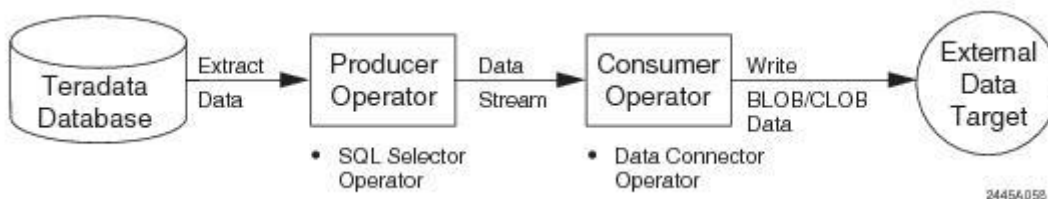
### Job Objective

Extract rows that include BLOB/CLOB/JSON data from a Teradata Database table and write them to an external flat file.

### Data Flow Diagram

Figure 41 shows a diagram of the elements for Job Example 14.

Figure 41: Job Example PTS00021, PS00027-- Extract BLOB/CLOB/JSON Data and Write It to an External File



## Sample Script

For the sample script that corresponds to this job, see the following scripts in the sample/userguide directory:

PTS00021: Extracting BLOB/CLOB Data and Writing It to an External Target.

**PTS00027:** Extracting BLOB/CLOB/JSON Data and Writing It to an External Target.

## Rationale

This job uses the operators shown for the following reasons:

☐ Use SQL Selector operator because it is the only operator that can read BLOB/CLOB/JSON data from a Teradata Database and write it to separate external data files. One data file stores data for one LOB/JSON column.

☐ Use DataConnector operator because it is the only operator that can write LOB/JSON data to an external file.

## Job Example 17: Extract Rows and Write Them to a Hadoop File

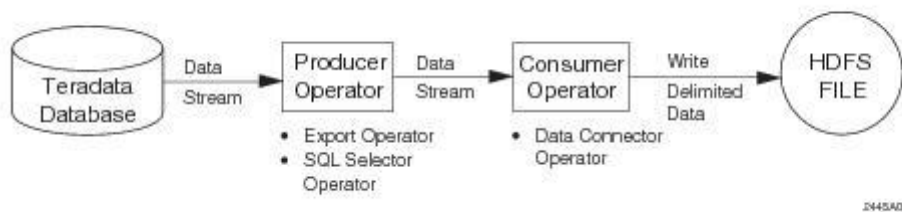
### Job Objective

This Teradata Parallel Transporter sample script loads five rows from a teradata table to flat file in Hadoop.

### Data Flow Diagram

Figure 42 shows a diagram of the elements for Job Example 17.

Figure 42: Job Example PTS00031-- Extract Rows and Write Them to a Hadoop File



## Sample Script

For the sample script that corresponds to this job, see the following script in the sample/userguide directory:

**PTS00031:** Extract Rows and Write Them to a Hadoop File

## Rationale

This job uses the operators shown for the following reasons:

- ❑ Export operator because it is the fastest way to extract large amounts of data from a Teradata Database table.
- ❑ DataConnector operator along with the HDFS Interface because it is the only producer operator that writes data to Hadoop HDFS.

### Job Example 18: Extract Rows and Write Them to a Hadoop Table

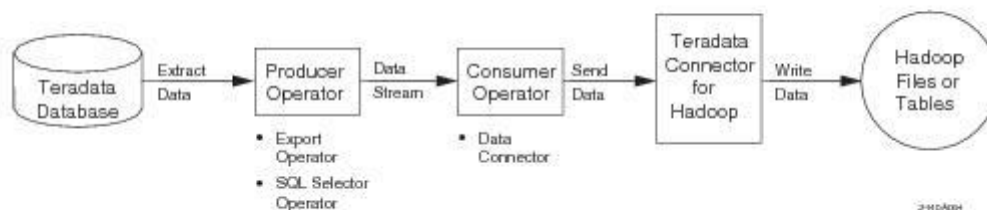
#### Job Objective

Extract rows from Teradata Database table and write them to a Hadoop table; the Hadoop table's data should be stored in the RCFile format.

#### Data Flow Diagram

Figure 43 shows a diagram of the elements for Job Example 18.

Figure 43: Job Example PTS00032- Extract Rows and Write Them to a Hadoop Table



#### Sample Script

For the sample script that corresponds to this job, see the following script in the sample/userguide directory:

PTS00032: Extract Rows and Write Them to a Hadoop Table

#### Rationale

This job uses the operators shown for the following reasons:

- ❑ Export operator because it is the fastest way to extract large amounts of data from a Teradata Database table.
- ❑ **DataConnector operator along with the TDCH-TPT Interface because it is the only producer operator**  
<https://docs.teradata.com/reader/b8dd8xEYJnxfsq4uFRrHQQ/uoSjTMZuwclBx9l5tG0B1Q>

#### Rules For Call Arguments In ODBC And JDBC in Vantage

The following additional rules apply to a call argument when the CALL statement is submitted from an ODBC or JDBC application:

- An IN or INOUT argument must be one of the following:
  - A value expression.

A value expression must not contain identifiers prefixed by the COLON character. It must be a constant expression.

- A QUESTION MARK (?) character used as an input placeholder.

If you specify ?, the value for the corresponding IN or INOUT parameter of the called procedure must be set using ODBC- or JDBC-specific calls prior to calling the procedure.

There is a 1:1 correspondence between the number of ? markers for IN and INOUT arguments and the number of data items specified in the StatementInfo parcel in the request message. StatementInfo does not contain entries for OUT arguments.

For example, consider the following SQL procedure definition and CALL statement:

```
CREATE PROCEDURE sp3 (  
  IN pi1  INTEGER,  
  INOUT pio1 INTEGER,  
  OUT po1  INTEGER)  
BEGIN  
  SELECT j INTO :pio1  
  FROM tb11  
  WHERE i=2;  
  SELECT k INTO :po1  
  FROM tb11  
  WHERE i=2;  
END;  
  
CALL sp3 (:?, :?, :?);
```

When this call is made, the StatementInfo parcel contains 2 entries: one each for the IN and INOUT parameters.

- An OUT argument must be an OUT call placeholder.

that writes data to Hadoop tables in the RCFile format.

<https://downloads.teradata.com/connectivity/articles/speed-up-your-jdbcodbc-applications>

Speed up your JDBC/ODBC applications

The Teradata JDBC Driver and ODBC Driver allow developers to quickly build applications that interact with the Teradata Database. However, many developers are surprised when their fully functioning application suddenly hits a performance roadblock when it is deployed to their production environment. And in many of these cases, the blame is sometimes unfairly placed onto the JDBC and ODBC drivers. This article will highlight the programming techniques available to maximize the performance when interacting with the database and help developers choose the right implementation.

### Quick and Easy but Slowest Performance

Many new database developers are more focused on how to create a database connection and pass a SQL statement than they are with performance. A typical first implementation looks something like:

```
1 Connection conn = DriverManager.getConnection(url, username, password);
2 Statement stmt = conn.createStatement();
3 String sql = "insert into Transactions(custID, transaction_date, amount, desc) values(" + custID + ", " + tran_date + ", " + a
4
5 stmt.executeUpdate(sql);
6
7 stmt.close(); // Your real code should use try-finally blocks to manage resources.
8 conn.close(); // Let's not even get into connection pools! That's another article.
```

Sure this works for a demo and the beginning programmer is probably pretty happy with the results. But turn on some production volume and this will quickly become a performance bottleneck, especially when your application is processing many SQL inserts such as when batch loading. This type of database coding is pretty much like driving your sports car and staying stuck in first gear!

### Drivers Prepare Your Statements

A much better approach is to use Prepared Statements. These will provide significantly better performance by first sending the database the outlines of the SQL statement using variable parameters in place of the actual data. The database prepares the execution steps of the SQL statement to optimize performance, and the prepared statement can then be used over and over again. This avoids recalculating the execution steps for each individual request, which is what happens in the first example.

```
1 // These are done once ...
```

```
2 Connection conn = DriverManager.getConnection(url, username, password);
3 String sql = "insert into Transactions(custID, transaction_date, amount, desc) values(?,?,?,?)";
4
5 PreparedStatement ps = conn.prepareStatement(sql);
6
7 // ... and these can be repeated many times with different values.
8 ps.setInt(1, custID);
9 ps.setDate(2, tran_date);
10 ps.setBigDecimal(3, amount);
11 ps.setString(4, desc);
12
13 ps.executeUpdate();
```

### Batch Ready

Prepared Statement batches take your performance to the next level. In addition to the benefits of reusing the Prepared Statement, batching your input values also reduces the number of round trips to the database. A batch size of roughly 5,000 to 10,000 works well for most applications. Using batches can be 10 to 40 times faster than the previous approach.

```
1 // These are done once.
2 Connection conn = DriverManager.getConnection(url, username, password);
3 String sql = "insert into Transactions(custID, transaction_date, amount, desc) values(?,?,?,?)";
4 PreparedStatement ps = conn.prepareStatement(sql);
5
6 for ( /* Loop through input values */ )
7 {
8     for ( /* Loop through a subset of the input values - the desired batch size */ )
9     {
10         ps.setInt(1, custID);
11         ps.setDate(2, tran_date);
```

```
12     ps.setBigDecimal(3, amount);
13     ps.setString(4, desc);
14     ps.addBatch(); // adds the row of input values to the batch
15 }
16
17 // This is done once per the desired batch size.
18 ps.executeBatch(); // sends all the batched rows to the database
19 }
```

### Full Speed Ahead

For loading truly huge amounts of data, JDBC FastLoad can provide even better performance. There are a couple of caveats, however. JDBC FastLoad can only insert data into an empty table, and JDBC FastLoad is only recommended for loading large amounts of data -- at least 100,000 rows total.

The nice thing is that your Java code doesn't need to change in order to use JDBC FastLoad. Your application uses the exact same Prepared Statement batches as in the previous example. Just add TYPE=FASTLOAD to your connection parameters, and the Teradata JDBC Driver will use JDBC FastLoad for particular SQL requests, if it can.

Note that the recommended batch size for JDBC FastLoad is much higher than for a regular SQL Prepared Statement batch, which means you may need to increase your JVM heap size. To get top-notch performance, you need to use a batch size of roughly 50,000 to 100,000. Using JDBC FastLoad can be 3 to 10 times faster than the previous approach.

<https://teradata-docs.s3.amazonaws.com/doc/connectivity/jdbc/reference/current/frameset.html>

### Troubleshooting

This chapter provides information for troubleshooting problems that can occur when using the Teradata JDBC Driver.

#### Socket Communication Failure

Error 804 with SQLState 08S01 and the error message "Socket communication failure for Packet receive" (or "Packet transmit") means that a network communication failure occurred.

[Error 804] [SQLState 08S01] Socket communication failure for Packet receive ...

[Error 804] [SQLState 08S01] Socket communication failure for Packet transmit ...

A network communication failure can occur due to a variety of reasons. Here is a list of common causes of connectivity problems, in order from most likely to less likely:

1. The Teradata session was forcibly logged off by Teradata Viewpoint, Teradata Manager, PMON, or some other administrator process that checks for session inactivity and aborts idle sessions. This can be checked by examining /var/log/messages on the database node, to look for messages that indicate that a session was aborted. This is a common problem for JDBC connections in a connection pool, because JDBC connections in a connection pool may spend a significant portion of their lifetime being idle. The database administrator should not forcibly log off idle Teradata sessions that are pooled JDBC connections, because that defeats the purpose of the JDBC connection pool.
2. Network problem and/or transient network failure. This can include situations such as a laptop switching from a wired to a wireless network connection (or vice-versa), or connecting to, or disconnecting from, a VPN.
3. Faulty network hardware, such as a faulty switch, router, or load balancer.
4. Database restart occurred. This can be checked by examining /var/log/messages on the database node, to look for messages that indicate that a database restart occurred.

### Numeric Data Truncation

Teradata Database V2R6.2 introduced support for the SQL data type BIGINT (64-bit integer) and introduced the Large Decimal feature, which expands the maximum precision for the DECIMAL data type to DECIMAL(38). Teradata Database V2R6.1 and earlier releases are limited to a maximum precision of DECIMAL(18).

Maximum precision varies by database release. This affects how numeric data is handled in the Teradata JDBC Driver. If Large Decimal is not supported, the maximum precision for BigDecimal is 18. If Large Decimal is supported, the maximum precision value is 38.

The Teradata JDBC Driver modification allows the PreparedStatement.setBigDecimal method to throw a DataTruncation exception for BigDecimal values that have precision values greater than the maximum precision.

When the PreparedStatement.setBigDecimal method is used to bind multiple values to a parameter, the Teradata JDBC Driver determines the largest number of integral digits bound to the parameter, and then the fractional digits for each of the values is rounded as necessary to fit within the database limit of maximum precision for a DECIMAL value. The method PreparedStatement.setLong in the Teradata JDBC Driver throws a DataTruncation exception if the maximum precision value is greater than 18 and the SQL data type BIGINT is not supported for the current database.

### Character Export Width

Using connection parameter CHARSET=UTF8 with fixed-width CHAR data type result set column values adds trailing space padding per the database's Character Export Width behavior. The CHAR(n) data type is a fixed-width data type (holding n characters), and the database reserves a fixed number of bytes for the CHAR(n) data type in response spools and in network message traffic.

UTF8 is a variable-width character encoding scheme that requires a varying number of bytes for each character. When the UTF8 session character set is used, the database reserves the maximum number of bytes that the CHAR(n) data type could occupy in response spools and in network message traffic. When the UTF8 session character set is used, the database appends padding characters to the tail end of CHAR(n) values smaller than the reserved maximum size, so that the CHAR(n) values all occupy the same fixed number of bytes in response spools and in network message traffic. In contrast, when using the UTF16 session character set, no character padding is added.

The following example illustrates how to work around this drawback by using CAST or TRIM in SQL SELECT statements, or in views, to convert fixed-width CHAR data types to VARCHAR.

- Given a table with fixed-width CHAR columns

```
CREATE TABLE MyTable (Column1 CHAR(10), Column2 CHAR(10))
```

- Original query

```
SELECT Column1, Column2 FROM MyTable
```

- Modified query using CAST and TRIM

```
SELECT CAST(Column1 AS VARCHAR(10)), TRIM(TRAILING FROM Column2) FROM  
MyTable
```

- View using CAST and TRIM

```
CREATE VIEW MyView (C1, C2) AS SELECT CAST(Column1 AS VARCHAR(10)),  
TRIM(TRAILING FROM Column2) FROM MyTable
```

Alternatively, connection parameter CHARSET=UTF16 is recommended for applications that require fixed-width CHAR data values without trailing space padding.

Transaction Isolation, Concurrency, and Deadlock

### Create and Drop

The following error may be seen when creating or dropping a database object, such as a table or stored procedure. It will include an error code of 2631 and an SQL state of 40001, which indicates that this is a retryable error:

```
com.teradata.jdbc.jdbc_4.util.JDBCException:[Teradata Database]: Transaction ABORTed due to deadlock.
```

If this error occurs, the application can choose to wait a short time and then resubmit the failed create or drop operation.

### JDBC FastLoad

The following error may be seen when using JDBC FastLoad and calling a PreparedStatement setter method. It will include an error code of 2631 and an SQL state of 40001, which indicates that this is a retryable error:

```
com.teradata.jdbc.jdbc_4.util.JDBCException:[Teradata Database]: Transaction ABORTed due to deadlock
```

If this error occurs, the application can choose to wait a short time and then call the PreparedStatement setter method again. Note that error 2631 may be in a chain of exceptions; it therefore is necessary to walk down the chain of exceptions to get to error 2631.

## Transaction Isolation

A potential deadlock condition can occur with two separate applications, or a single application using two threads, with each thread or application having its own JDBC connection to the database.

The problem occurs when one connection is inserting data into a table, while the other connection is attempting to read data from the same table.

When using the default transaction isolation level of TRANSACTION\_SERIALIZABLE, the following error may be seen on the thread or application that is reading from the table, approximately 2 to 5 minutes after the situation occurs. It includes an error code of 2631.

com.teradata.jdbc.jdbc\_4.util.JDBCException:[Teradata Database]: Transaction ABORTed due to deadlock.

If this error occurs, either:

- Resubmit the failed read operation, or
- Use a transaction isolation level of TRANSACTION\_READ\_UNCOMMITTED on the connection reading from the table.

**Note:** The transaction level is set using the java.sql.Connection.setTransactionIsolation method. Though this prevents the problem from occurring, it has the side effect of allowing dirty, non-repeatable, and phantom reads. Whether or not this is acceptable must be determined on an individual application basis.

## Improving Performance

If the performance of the application seems very slow, here are some recommendations for improvement:

- **Turn off debug parameters.** Make sure all debugging is turned off. See [Using the Teradata JDBC Driver](#).
- **Use PreparedStatement where possible.** This applies whenever the same SQL statement is submitted many times, but data values differ for each submission. One example would be an INSERT statement that is submitted many times, but with different inserted data values each time. Another example would be a SELECT statement that is submitted many times, but with different comparison values in WHERE-clause conditions each time.

If data values are specified as literals in the SQL statement, and the SQL statement is changed with different literal data values upon each submission, then the database must parse the SQL statement each time before executing it.

For situations like these, a PreparedStatement should be used instead. The SQL statement must have a ? placeholder for each data value that will be changed per submission.

The application must prepare the SQL statement once, using the `Connection.prepareStatement` method. For each submission, the application must bind all the data values using the `PreparedStatement.setXXX` methods, and then the application must execute the `PreparedStatement`.

The application can repeat the bind and execute steps over and over, with different bound data values each time. This technique provides a substantial performance improvement, because the database only needs to parse the SQL statement once, and can re-execute the parsed statement over and over.

- **Inserting Small LOB Values.** The recommended technique for inserting LOB values is to use a `PreparedStatement INSERT` with `?` parameter markers for all column values to be inserted. Use the `setBinaryStream` method for binding BLOB values to the parameter markers corresponding to BLOB columns, then use the `setAsciiStream` or `setCharacterStream` method for binding CLOB values to the parameter markers corresponding to CLOB columns.

When the `setBinaryStream`, `setAsciiStream`, and `setCharacterStream` methods are used, the Teradata JDBC Driver sends LOB data to the database separately from other bound parameter values, so that LOB values do not count towards the database limit on total bytes of bound parameter values per inserted row.

To improve the performance of a `PreparedStatement INSERT`, that is inserting one or more small ( $\leq 64000$  bytes) LOB values per row, the `setString` method is used to bind a value to a CLOB column, and the `setBytes` method is used to bind a value to a BLOB column. The SQL `INSERT` statement must cast the `?` parameter marker to a CLOB or BLOB, respectively.

```
INSERT INTO MyTable(id,clob_col) VALUES(?,CAST(? AS CLOB))
prepStmt.setInt(1,id);
prepStmt.setString(2,"abc");
```

Using the `setBytes` method with a `CAST` expression forces the Teradata JDBC Driver to send the bound parameter value as a `VARBYTE` value, so it is limited to 64000 bytes, even though the destination column may be a BLOB that can hold values larger than 64000 bytes.

Using the `setString` method with a `CAST` expression forces the Teradata JDBC Driver to send the bound parameter value as a `VARCHAR` value, so it is limited to 64000 bytes, even though the destination column may be a CLOB that can hold values larger than 64000 bytes. If a Unicode session character set (UTF8 or UTF16) is used, and/or if the destination column is designated `CHARACTER SET UNICODE`, then the database will convert the bound parameter value into two-byte Unicode characters. The value after conversion is limited to 64000 bytes.

This technique works only if the total size of all the bound parameter values does *not* exceed the database limit on total bytes for all the bound parameter values for an inserted row. This technique should only be used when performance is critical, and it is known in advance that the total size of all the bound parameter values, including LOB values, does *not* exceed 64000 bytes per inserted row.

This technique is subject to a further limitation such that the total size of all the bound parameter values must not exceed the database limit on total bytes for all the bound parameter values for an inserted row, after any necessary character set conversions have been performed by the database. If a Unicode session character set (UTF8 or UTF16) is used, and/or if a destination character column is designated CHARACTER SET UNICODE, then the database will convert all the bound parameter values that are character data types (CHAR, VARCHAR, CLOB) into two-byte Unicode characters. These two-byte Unicode characters are counted towards the database limit on total bytes for all the bound parameter values for an inserted row.

- **Use executeBatch() where possible.** Whenever there are many insert, update, or delete statements that can be submitted together, use the executeBatch() method rather than executeUpdate() or execute(). However, the total buffer length is limited to approximately 1 MB. Using executeBatch() instead of executeUpdate() can improve your performance by more than 50%.
- **Use connection pooling provided by an application server.** Connection pooling is a technique used for sharing server resources among requesting clients. It allows multiple clients to share a cached set of connection objects that provide access to the database. It improves performance by eliminating the overhead associated with establishing a new database connection for each request. However, there are some restrictions. Since it is not currently possible to reset a database connection, users of connection pooling must not change the following session parameters because these changes will be inherited by the next user of the connection:
  - Database
  - Collation
  - Character Set
  - Transaction Semantics
  - Dateform
  - Timezone
  - Default Date Format
- **Use multi-threading.** Where possible, use multi-threading with multiple sessions for those requests that can be processed at the same time. It is important to remember, however, not to have multiple concurrent requests on a single session. Teradata does not support this and even though the driver will accept this, it blocks until the current request is complete. This may actually degrade performance. For improved performance, use concurrent sessions with each session running only one request at a time.
- **Use a Transaction isolation level of TRANSACTION\_READ\_UNCOMMITTED.** This feature can speed up access to data though it comes at the cost of encountering dirty reads, non-repeatable reads, and phantom reads. Whether or not this is suitable should be determined on an individual application basis.
- **Use TYPE\_SCROLL\_INSENSITIVE result sets.** These can improve performance when used with queries which can return large multiple result sets that do not require all rows to be processed.

Beginning with Teradata Database 12.0, when the application requests the ResultSet type to be `ResultSet.TYPE_SCROLL_INSENSITIVE`, the Teradata JDBC Driver is able to quickly and efficiently skip to the next result of a multi-statement request by using cursor positioning to position to the last row of the current result set. If forward-only result sets are used, the same skipping operation will require the JDBC driver to fetch all rows of the current result set first, which can take significantly longer.

The following methods will create statements that return `TYPE_SCROLL_INSENSITIVE` result sets:

```
Connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_READ_ONLY)  
Connection.prepareStatement(sql, ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_READ_ONLY)  
Connection.prepareCall(sql, ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_READ_ONLY)
```

- **Use a single database hostname in DNS.** Improve connection time with the following steps.
  - Define a single DNS name with multiple IP addresses for the database, such that each IP address corresponds to a database node running a DBS Gateway. Omit the IP addresses of nodes that don't normally run a DBS Gateway, such as Hot Standby Nodes in most configurations. Enable DNS round-robin for the IP addresses, so that connections are distributed across all the nodes.

- Specify `COP=OFF` for Teradata JDBC Driver connections

This avoids the time-consuming COP discovery process. The Teradata JDBC Driver will attempt to connect to the first IP address returned by the DNS lookup, and will use subsequent IP addresses in case of a connection failure. Use DNS round-robin to distribute Teradata JDBC Driver connections across all available nodes.

<https://docs.teradata.com/reader/3AkrVQlhjJMha4KRVJmm1w/Qwm4xSjCLk6tol7XOMnMPA>

## Analytic Tools

### Teradata AppCenter

Teradata AppCenter is a self-service execution platform for creating and running SQL and Basic Teradata Query (BTEQ) applications (apps). You can query data and run jobs automatically or on demand based on selected schedule options. AppCenter includes privacy settings for both apps and job results.

This self-service environment enables the easy creation and reuse of analytics. It is composed of numerous pre-built features that allow data scientists and developers to build, share, and deploy analytics with AppCenter. Non-technical users can run apps, visually study results, and share insights. The apps are easily accessed and deployed on-premises or in the cloud.

<https://docs.teradata.com/reader/B7Lgdw6r3719WUyiCSJcgw/ U8FBA1PgnQMYN0mWs8oNQ>

## Workload connections of tools through Account Strings Expansion

### Considerations for Assigning ASE Variables to Different Workloads

Each ASE assignment depends on the type of usage being performed by the user ID. This has implications related to user ID assignment.

In general, workloads can be broadly grouped into three categories as follows:

#### ☐ Multisession/Multirequest

This workload can be identified by work typically done by MultiLoad, FastLoad, TPUMP or multisession BTEQ. These types of activities are normally used for database maintenance. Each session used handles multiple requests over time. The workload for this type of work tends to be more predictable and stable. It runs regularly and processes the same way each time it runs.

#### ☐ Single Session, nontactical

This workload is typically initiated through a single session BTEQ, SQL Assistant, MicroStrategy, or other query-generating tool. Ad hoc users, or single session BTEQ jobs in the batch process can generate this type of activity. The single session may generate one or many requests. The requests may be back to back, or there may be hours of idle time between them. Typically, the requesting user has very broad and informal response time expectations.

#### ☐ Single Session, tactical

This workload is similar to the Single Session workload category except that there is typically a very clear definition of response time and the response time requirements normally range between less than a second to a few seconds.

The following ASE variables are to be used for each of the workload categories listed above along with the rationale for selecting the ASE variables.

#### ☐ Multisession, Multirequest

For this workload, usage information need not be captured at the request level. Workloads in this category can

☐ Processes the same request over and over again across the multiple sessions it establishes (such as TPUMP and multisession BTEQ).

☐ Generate multiple internal requests that are not easily correlated to specific user generated activity (as is the case with MultiLoad and FastLoad).

As a result, capturing usage detail at the request level typically does not provide especially meaningful information. Therefore, the recommended standard is to capture usage at the session level using the '&S' ASE variable. The account string for User Ids performing this workload category would have the following format:

Account String Format: \$XX\$\_&S

Length: 12-15 characters (depending on PG length)

Capturing session level information for this workload category provides several benefits, including:

- ☐ All usage for a given job can be more easily captured. Furthermore, the job level usage can then be grouped to associate all batch processing to an application.
- ☐ All usage for a given job step can be obtained. This can facilitate performance analysis for batch processes.
- ☐ Session usage within a multisession utility can be better analyzed to determine the optimal number of sessions to log on to the system.
- ☐ Single Session, nontactical

For this workload, request level usage detail is desired. This type of activity is typically the most difficult to manage and control in a mixed workload, data warehouse environment. They also typically represent the greatest opportunity for optimization. Although request level detail requires some minor additional overhead to capture, the benefits of gaining additional visibility into the impact of each request outweighs the increased overhead in data collection. The account string for user IDs performing this workload category would have the following format:

Format: \$XX\$\_&I

Length: Up to 128 characters

Capturing request level information in this manner has numerous benefits, including:

- ☐ Usage associated with each SQL request can be identified. By applying specific metrics such as total CPU used, total IO used, CPU skew percent, Disk to CPU ratio, and so forth, problem requests can quickly and easily be identified and addressed.
- ☐ Request level usage detail can be correlated to SQL statements in DBQL to greatly simplify performance-tuning efforts. DBQL captures the date and time of the request as well as the session and request number of the request.
- ☐ Performance tuning can become much more quantitative and definitive by comparing usage statistics for alternative query approaches. Capturing the consumption at the individual request enables this benefit.
- ☐ Usage can be accumulated to the session level to provide same level aggregations and analysis to multisession, multirequest processing. As such, the same benefits can also be achieved.
- ☐ Single Session, tactical

For this workload, high-speed performance and minimal response time are the primary objectives. Even if the Teradata Active EDW is not currently servicing this type of request, it is important to account for this type of work within the standard. Typically, this workload tends to be very predictable in nature with queries typically designed to be single AMP retrievals. For this workload, capturing information at the request level is unnecessary for two reasons. First, the transactions are well defined and repeated over and over again. Second, the additional overhead required to record usage for each request would represent a meaningful portion of the overall work performed on behalf of the transaction. In other words, the additional overhead could materially impact request response time.

As a result, the account string for this workload can, as one option, target usage detail at the session level. The assumption in this case is that applications requiring high-volume, low response time requests will take advantage of session pooling to avoid the overhead of continually logging on and logging off. The account string for User Ids performing this workload category would have the following format:

Format: \$XX\$\_&S

Length: 12-15 characters (depending on PG length)

Since this is the same ASE strategy as employed for the multisession, multirequest workload, all the same benefits would accrue. In addition, as it pertains to this particular workload category, the following benefits could also be achieved:

- ☐ Usage by session could assist in determining the optimal number of sessions to establish for the session pool.
- ☐ CPU and/or IO skew by session could help identify possible problems in the data model for the primary index retrievals.

#### About Using ASE With Client Utilities

Except for the utilities and variables noted in the table that follows, you can use ASE with any utility that uses a standard Teradata Database interface to log on, including:

- ☐ BTEQ
- ☐ FastLoad
- ☐ MultiLoad
- ☐ Teradata Parallel Transporter
- ☐ TPump (except for &T)
- ☐ FastExport
- ☐ Teradata SQL Assistant (formerly known as Queryman)
- ☐ Teradata Studio

[https://teradata.sharepoint.com/:w:/r/sites/COMPAS/\\_layouts/15/doc2.aspx?sourcedoc=%7B18CB7D7E-61E9-4B32-867D-F9B2042C8D92%7D&file=Power%20BI%20Overview%20-%20DA013199.docx&action=default&mobileredirect=true&DefaultItemOpen=1&cid=8faaa9fa-7cf9-48e8-9616-2c09999bb1e3](https://teradata.sharepoint.com/:w:/r/sites/COMPAS/_layouts/15/doc2.aspx?sourcedoc=%7B18CB7D7E-61E9-4B32-867D-F9B2042C8D92%7D&file=Power%20BI%20Overview%20-%20DA013199.docx&action=default&mobileredirect=true&DefaultItemOpen=1&cid=8faaa9fa-7cf9-48e8-9616-2c09999bb1e3)

( Section 9)

## POWER BI CONNECTIONS

### DirectQuery vs Live Connection vs Imported Data

DirectQuery refers to relational data sources. Live Connection refers to Analysis Services sources. Though the terms differ, they both represent the same type of functionality – Live Connectivity – where the source data remains in the source.

Imported Data means the source data is replicated, or imported, into a data model stored in Power BI. Data refresh operations are required for the data to remain current.

#### 9.1 Live Connectivity

Live connectivity is best for the following situations:

1. The source data is complete and does \*not\* need to be augmented with additional data sources – for instance, traditional data warehousing.
2. Near real-time (low latency) data is required.
3. Data is updated frequently in the source (and a secondary data refresh is not desired).
4. Corporate security standards dictate the data may \*not\* be replicated into another data source.
5. Higher data volumes are involved which exceed the 250 MB limit of a Power BI embedded data model.
6. Row-level security is centralized in an SSAS Tabular model or underlying data source.

NOTE: On-Prem data connectivity requires the enterprise gateway.

#### 9.2 Imported data

Imported data is most suitable for the following situations:

1. Existing data is to be augmented with additional data sources (such as industry data, demographics, weather, etc). This is frequently referred to as data mashups.
2. Additional calculations are required that do not exist in the data source.
3. Exploratory reporting scenarios, prototyping activities, and one-time projects.

4. The data can fit into 250 MB (compressed), the max size for an embedded data model.
5. It is appropriate for row-level security to be specified for one specific data model in the Power BI Service. From a governance standpoint, specifying row-level security for a single model is riskier than utilizing a centralized source.

NOTE: On-Prem data requires either the personal or enterprise gateway to keep data current.

DirectQuery refers to relational data sources. Live Connection refers to Analysis Services sources. Though the terms differ, they both represent the same type of functionality – Live Connectivity – where the source data remains in the source.

Imported Data means the source data is replicated, or imported, into a data model stored in Power BI. Data refresh operations are required for the data to remain current.

### 9.1 Live Connectivity

Live connectivity is best for the following situations:

1. The source data is complete and does *\*not\** need to be augmented with additional data sources – for instance, traditional data warehousing.
2. Near real-time (low latency) data is required.
3. Data is updated frequently in the source (and a secondary data refresh is not desired).
4. Corporate security standards dictate the data may *\*not\** be replicated into another data source.
5. Higher data volumes are involved which exceed the 250 MB limit of a Power BI embedded data model.
6. Row-level security is centralized in an SSAS Tabular model or underlying data source.

NOTE: On-Prem data connectivity requires the enterprise gateway.

### 9.2 Imported data

Imported data is most suitable for the following situations:

1. Existing data is to be augmented with additional data sources (such as industry data, demographics, weather, etc). This is frequently referred to as data mashups.
2. Additional calculations are required that do not exist in the data source.
3. Exploratory reporting scenarios, prototyping activities, and one-time projects.
4. The data can fit into 250 MB (compressed), the max size for an embedded data model.

5. It is appropriate for row-level security to be specified for one specific data model in the Power BI Service. From a governance standpoint, specifying row-level security for a single model is riskier than utilizing a centralized source.

NOTE: On-Prem data requires either the personal or enterprise gateway to keep data current

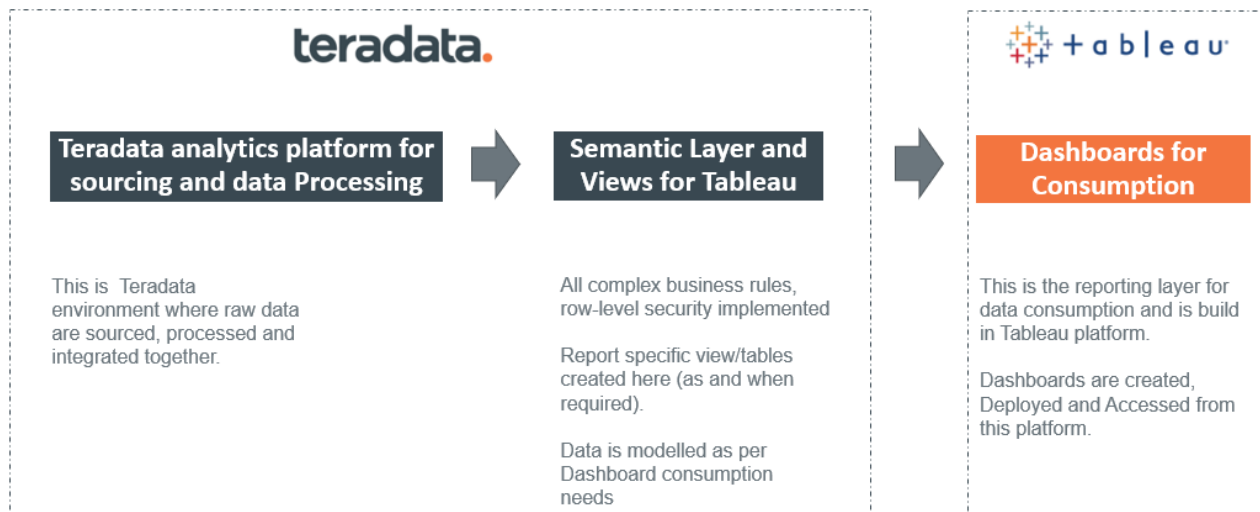
[https://teradata.sharepoint.com/:p:/r/sites/COMPAS/\\_layouts/15/Doc.aspx?sourcedoc=%7B9E757331-019D-4D9C-A0B4-0E7D51B699EF%7D&file=Tableau%20Teradata%20Best%20Practices%20-%20DA010806.pptx&action=edit&mobileredirect=true&DefaultItemOpen=1&cid=ad1564b2-06d0-45a1-9384-fb81a7efb9e7](https://teradata.sharepoint.com/:p:/r/sites/COMPAS/_layouts/15/Doc.aspx?sourcedoc=%7B9E757331-019D-4D9C-A0B4-0E7D51B699EF%7D&file=Tableau%20Teradata%20Best%20Practices%20-%20DA010806.pptx&action=edit&mobileredirect=true&DefaultItemOpen=1&cid=ad1564b2-06d0-45a1-9384-fb81a7efb9e7)

( Slide 7, Slide 8)

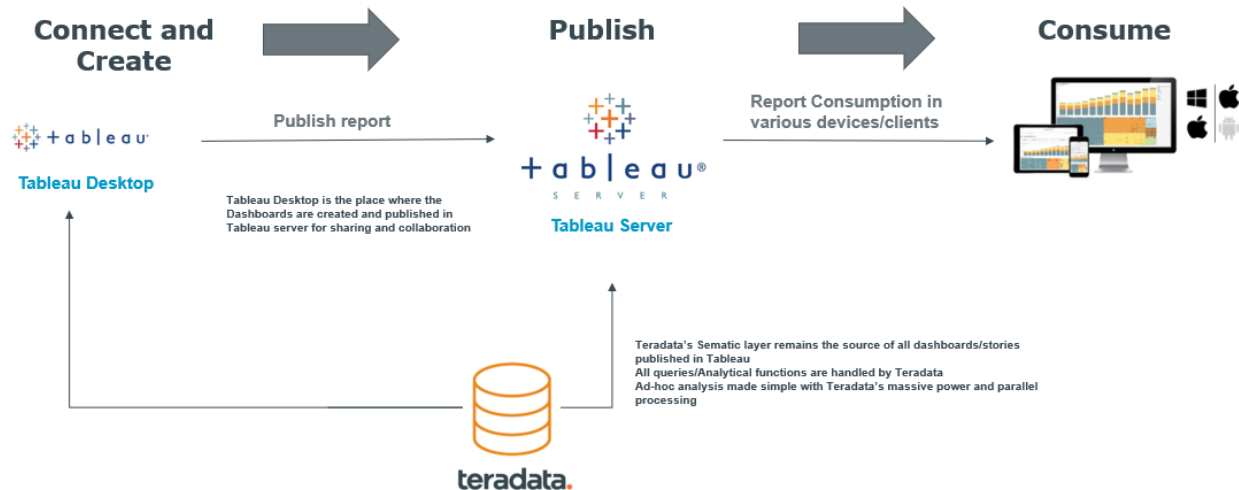
and Teradata Performance Tuning Best Practices (High Level)

### Teradata- Tableau Standard data flow diagrams

Below diagram is a high-level representation of standard data flow in Teradata-Tableau integrated platform:



Data flow is further broken down into Tableau components:



## Teradata Performance Tuning Tips For Tableau

### Indexes

- Validate indexes (eliminate skew due to zeroes or nulls)
- Create secondary indexes on dimension fields

Data Compression/ Multi value compression (MVC) on low cardinality columns

Data Distribution

Collect Statistics

Semantic Layer Views

- ✓ Embed joins and business logic in views to minimize complexity in report queries, use join indexes,
- ✓ De-normalize the physical database design as and when necessary, where the associated costs are justified by perceived benefits for the business
- ✓ Can be leveraged by multiple BI tools

Implement Aggregate Join Indexes to provide quicker response time and lessen resource usage for often executed user queries that aggregate millions of rows

## Detailed Performance Tuning

This section includes the detail of various performance tuning checkpoints and the best practices to implement the tuning.

### Performance Tuning checkpoints in Teradata:

1. Database Integrity Constraints
2. Aggregate Join Indexes
3. Indexing:
  - Validate Primary Indexes (Eliminate skew due to zeros / nulls)
  - Secondary indexes on Dimensions, Join and Hash Indexes

4. Data Compression/ Multi value compression (MVC) on low cardinality columns
5. Data Distribution
6. Collect Statistics
7. Use Query logs to analyze Tableau queries inside database
8. Use Query Bands captured in tableau queries for further tuning

#### Performance Tuning checkpoints in Tableau:

1. Dashboard and Worksheets
2. Filtering
3. Summary Tables
4. Queries
5. Extracts
6. Performance Recording

#### Teradata features supported by Tableau:

##### Teradata Performance Tuning Best Practices (High Level)

This section lists some of the best practices (with short description) followed for the purpose of optimizing BI queries and operation in Teradata. However, optimization is a scenario-based requirement and every separate implementation would require specific tuning appropriate for the same.

1. Teradata recommends that a combination of entity and referential integrity constraints be used in an Enterprise Data Warehouse environment to support business user analytic access
2. Teradata recommends the use of Aggregate Join Indexes to provide quicker response time and lessen resource usage for often executed user queries that aggregate millions of rows.
3. Appropriate Indexing:

##### **Primary Indexes**

- Teradata Database uses the Primary indexes as a distribution index that must be chosen based on followed considerations:
  - To maximize one-AMP operations
  - To optimize parallel processing
  - To minimize expensive I/O operations
- Ideally, Primary Index should be created as a selection of column(s) that most often used to access the data and column(s) with stable data value. No more than 64 columns can be specified in a primary index definition.
- As Primary Index can be unique or non-unique, note that then more unique the PI that better the distribution of values.
- Only one primary index can be defined on a table and a single-value Primary Index access requires only one AMP and typically and disk I/O.

##### **Secondary Indexes**

- Secondary Indexes provide applications using an alternate access path with the better

performance. Secondary Indexes are optional and may be:

- Unique or non-unique
- Dynamically created and dropped by the user for optimum performance, according to application requirements
- Do not affect a table distribution and can reduce base table I/O during value and join operations
- Secondary Indexes are recommended on dimensions and master tables to provide enhanced performance on drilldowns
- The query workload should be analyzed regularly. Teradata Index Wizard should be used to advise which indexes should be created to benefit query performance and decrease overall resource usage
- The Teradata Index Wizard also identifies indexes that are being underutilized and can be dropped so that an optimal number of indexes exist as query workloads change over time.

NoPI

- A new type of Teradata table without a Primary Index (PI). This type of table is neither hash distributed, nor hash ordered and allows rows to be appended to the end of a table as if it were a spool file.
- Tableau may consider supporting this feature to ensure the quick creation of small temporary tables necessary for certain types of analytical queries.

PPI:

- Partitioned Primary Indexes (PPIs) is partitioning and indexing mechanism used in physical database design.
- When using PPIs (single-level or multi-level):
  - Multiple levels of partitioning are allowed (but beware of over partitioning).
  - Rows are still hash distributed among the AMPs on the primary index's columns
  - Rows are ordered by the first level partitioning, then second, etc. and finally by the hash value of the primary index columns.
  - In many cases, better performance occurs when the partitioning column are part of the primary index; however, adding columns to the primary index may reduce the usefulness of the primary index for access, joins, and aggregations.
  - PPIs are especially helpful for queries based on range access, such as date ranges.
  - PPIs are allowed for volatile tables or global temporary tables since V2R6.1, so the SQL Engine automatically generates partitioned primary indexes when creating temporary tables and provides enough control for performance tuning.

For more information, refer to the Orange Book: Partitioned Primary Index Usage (Single-Level and Multilevel Partitioning (541-0003869-E02)

#### 4. *Multi-Value Compression should also be used on low cardinality columns.*

- Multi-value Compression (MVC) should be used on low cardinality columns or where a small number of value (255 or less) occur very frequently compared to other values.
- Algorithmic Compression (ALC) should be used on wide values such as character strings where there are usually effective algorithms to compress the data; however, consider the trade-offs of less space usage and reduced I/O vs. the CPU overhead of compressing/decompressing the data.

- Block-level compression (BLC) can also be used to reduce space usage and I/O with the trade-off of increased CPU overhead to compress and decompress the data blocks. Temperature-based BLC can be used to reduce the impact of CPU overhead by applying the compression automatically to cold data while leaving hot data uncompressed (I am not sure which release temperature-based BLC was added so if the BP is for a particular release or set of releases you need to check this).

Refer to Orange Book "Compression in Teradata 13.10" 541 – 0008669 – A02

5. *The data distribution strategy in Teradata is crucial to good performance*

- Teradata is a parallel database system and the data is distributed across a set of AMPs. Each AMP performs all database work such as reading, updating, sorting, journaling, locking and

indexing. For maximum performance, it is important to design the database to evenly distribute the data across AMPs to ensure that all AMPs in the system perform approximately the same amount of work.

Any user tables joined to EDW tables need to be aware of how their data is distributed and how that of the EDW tables is used? Large EDW tables joined together either need to have their data distributed the same way or join indexes created to meet usage requirements

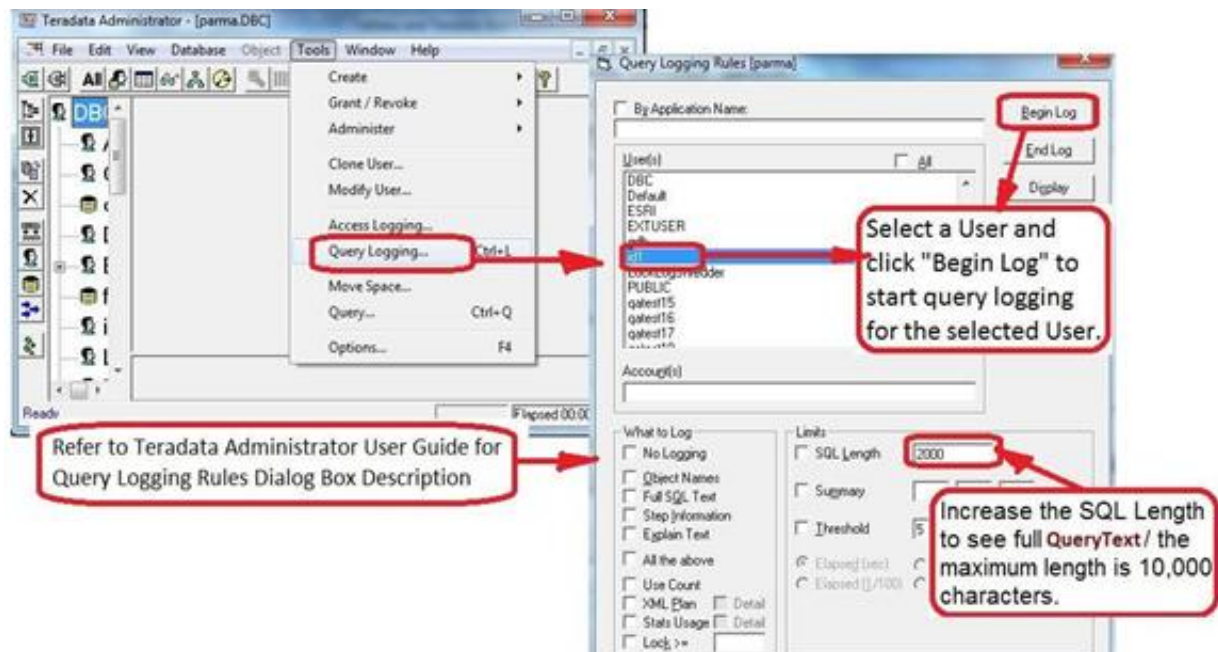
A limited amount of de-normalization should be used where it is beneficial to reduce the number of joins for performance reasons and to simplify query optimization

6. *The need for collection of statistics and keeping statistics fresh.*

- In most cases, collecting statistics will help improve query performance. With Collected Statistics the Teradata optimizer does not allow the user to “override” its decisions through constructs like hints or rewritten SQL. The optimizer will select the best plan to access the data.
- Tips to consider: Do not collect stats if it is not improving your query. This is just adding more process without any benefit. Use Explain <sql statement> and compare the plan, the cpu and the IO. Note: elapsed time and the time is not a good comparison method.
  - You may not need to collect stats of more than 3 or 4 columns together (multi column stats).
  - Start with single column stats recommendation from the diagnostic helpstats plus a few multicolumn stats that says high confidence, if they are less than 3 or 4 columns. The number of columns in the multicolumn stats depends on the size (in bytes) of each column. The total should not be greater than 16 bytes, which is usually 4 columns of integer data type.
- Remove unwanted and stale statistics.
- Please refer to the following documentation for Best Practices for Teradata Statistics Collection: Orange Books:
  - Automated Statistics Management Teradata 14.10 (541-0009628-A03)
  - Teradata 14.0 Statistics Enhancements 2011-12 (541-0009064-A02)
  - Statistics Extrapolations 2010-09 (541-0008668-A02)
  - Collecting Teradata Statistics 2007-03 (541-0006463-A02)

## 7. Using DBQL to analyze Tableau queries inside database

- With Teradata Administrator, from the main window, click Tools>Query Logging.



## Extracts (Tableau)

- Persistent cache of data that is written to disk and reproducible
- Columnar data store - a format where the data has been optimized for analytic querying
- Completely disconnected from the database during querying
- Refreshable, either by completely regenerating the extract or by incrementally adding rows of data to an existing extract
- Architecture-aware – unlike most in-memory technologies it is not constrained by the amount of physical RAM available
- Portable – extracts are stored as files so can be copied to a local hard drive and used when the user is not connected to the corporate network. They can also be used to embed data into packaged workbooks that are distributed for use with Tableau Reader;

Often much faster than the underlying live data connection

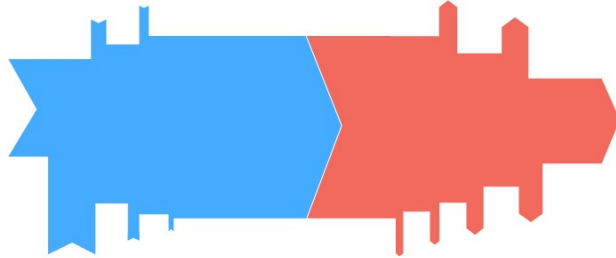
## Extracts vs. Live Connection(Tableau)

- The speed of the Tableau fast data engine is relative. You must consider the source data and the processing power you have already given that data, as well as the processing power you have on Tableau before you can determine whether the data engine is going to offer an improvement.
- For a non-optimized database or file-based data source, the data engine's processing is much faster and will result in a better user experience. But a well optimized database with indexing might be just as fast as the Tableau data engine.

At the other extreme, the Tableau data engine will probably be slower than a big cluster of machines like you would find with Data Warehouse appliances such as Teradata. You might create an aggregated extract to offload summary-style analysis, but then drill back to the detailed source data (using actions or blending) which would remain in the DW

Source: <https://datavizcatalogue.com/>

## Sankey Diagram

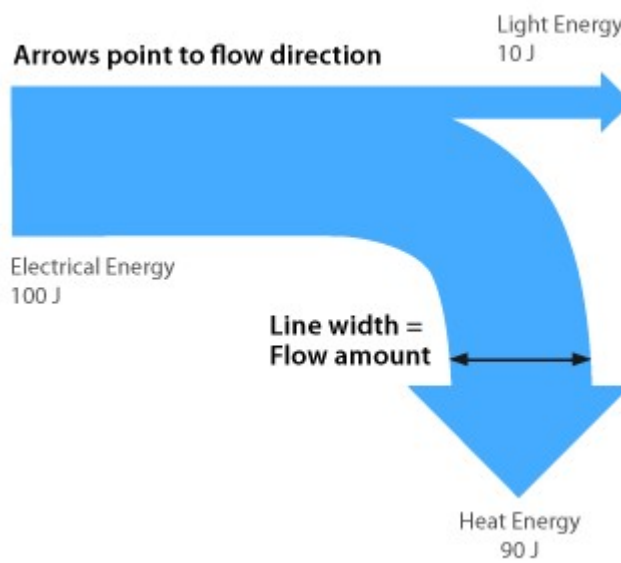


### Description

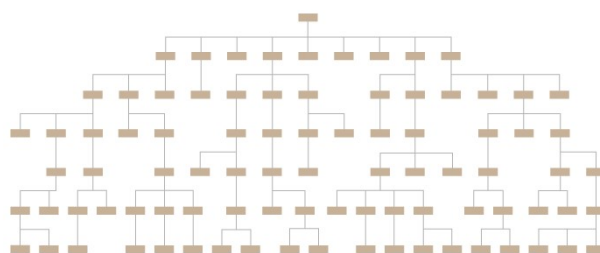
Sankey Diagrams display flows and their quantities in proportion to one another. The width of the arrows or lines are used to show their magnitudes, so the bigger the arrow, the larger the quantity of flow. Flow arrows or lines can combine together or split through their paths on each stage of a process. Colour can be used to divide the diagram into different categories or to show the transition from one state of the process to another.

Typically, Sankey Diagrams are used to visually show the transfer of energy, money or materials, but they can be used to show the flow of any isolated system process.

### Anatomy



# Tree Diagram



## Description

Also known as a *Organisational chart*, *Linkage Tree*.

A Tree Diagram is a way of visually representing hierarchy in a tree-like structure. Typically the structure of a Tree Diagram consists of elements such as a **root node**, a member that has no superior/parent. Then there are the **nodes**, which are linked together with line connections called **branches** that represent the relationships and connections between the members. Finally, the **leaf nodes** (or end-nodes) are members who have no children or child nodes.

Tree Diagrams are often used:

- To show family relations and descent.

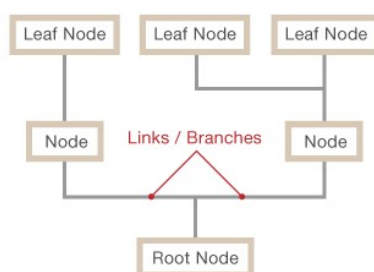
- In taxonomy, the practice and science of classification.

- In evolutionary science, to show the origin of species.

- In computer science and mathematics.

- In businesses and organisations for managerial purposes.

## Anatomy



# Pie Charts



## Description

Extensively used in presentations and offices, Pie Charts help show proportions and percentages between categories, by dividing a circle into proportional segments. Each arc length represents a proportion of each category, while the full circle represents the total sum of all the data, equal to 100%.

Pie Charts are ideal for giving the reader a quick idea of the proportional distribution of the data. However the major downsides to pie charts are:

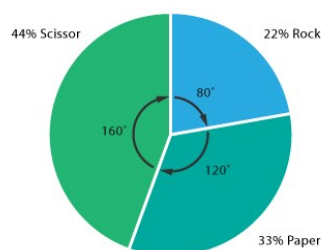
They cannot show more than a few values, because as the number of values shown increases, the size of each segment/slice becomes smaller. This makes them unsuitable for large amounts of data.

They take up more space than their alternatives, like a [100% Stacked Bar Chart](#) for example. Mainly due to their size and for the usual need for a legend.

They are not great for making accurate comparisons between groups of Pie Charts. This being that it is harder to distinguish the size of items via area when it is for length.

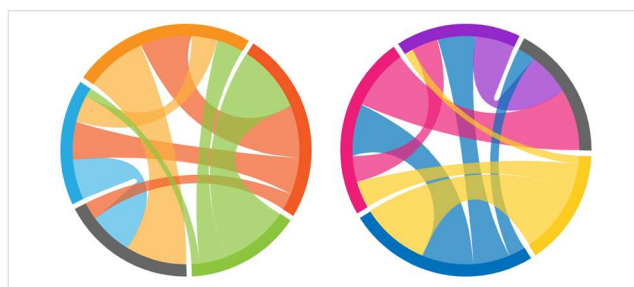
In spite of that, comparing a given category (one slice) within the total of a single Pie Chart, then it can often be more effective.

## Anatomy



Data			
Rock	Paper	Scissor	TOTAL
2	3	4	9
To calculate percentages			
$2/9=22\%$	$3/9=33\%$	$4/9=44\%$	100%
Degrees for each "pie slice"			
$(2/9) \times 360 = 80^\circ$	$(3/9) \times 360 = 120^\circ$	$(4/9) \times 360 = 160^\circ$	360°

# Chord Diagram



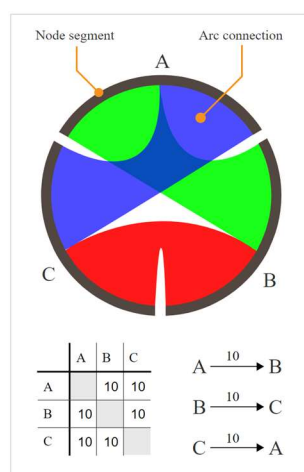
## Description

This type of diagram visualises the inter-relationships between entities. The connections between entities are used to display that they share something in common. This makes Chord Diagrams ideal for comparing the similarities within a dataset or between different groups of data.

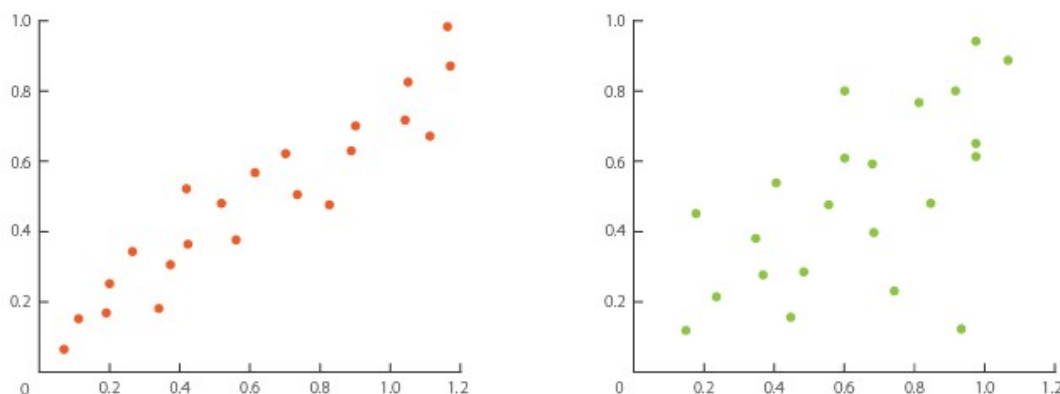
Nodes are arranged along a circle, with the relationships between points connected to each other either through the use of arcs or Bézier curves. Values are assigned to each connection, which is represented proportionally by the size of each arc. Colour can be used to group the data into different categories, which aids in making comparisons and distinguishing groups.

Over-cluttering becomes an issue with Chord Diagrams when there are too many connections displayed.

## Anatomy



# Scatterplot



## Description

Also known as a *Scatter Graph*, *Point Graph*, *X-Y Plot*, *Scatter Chart* or *Scattergram*.

Scatterplots use a collection of points placed using Cartesian Coordinates to display values from two variables. By displaying a variable in each axis, you can detect if a relationship or correlation between the two variables exists.

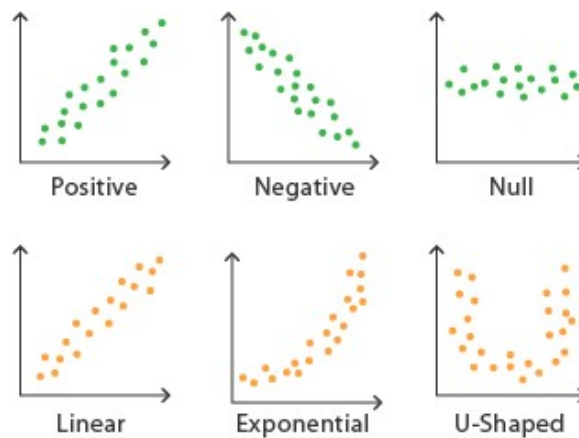
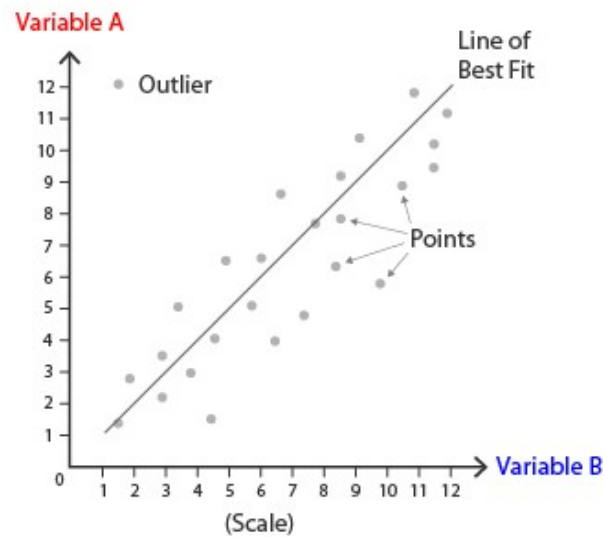
Various types of correlation can be interpreted through the patterns displayed on Scatterplots. These are: **positive** (values increase together), **negative** (one value decreases as the other increases), **null** (no correlation), **linear**, **exponential** and **U-shaped**. The strength of the correlation can be determined by how closely packed the points are to each other on the graph. Points that end up far outside the general cluster of points are known as **outliers**.

Lines or curves are fitted within the graph to aid in analysis and are drawn as close to all the points as possible and to show how all the points were condensed into a single line would look. This is typically known as the **Line of Best Fit** or a **Trend Line** and can be used to make estimates via interpolation.

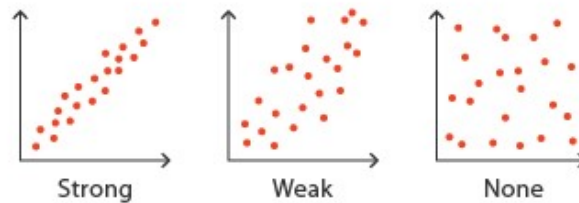
Scatterplots are ideal when you have paired numerical data and you want to see if one variable impacts the other. However, do remember that correlation is not causation and another unnoticed variable may be influencing results.

# Scatterplot

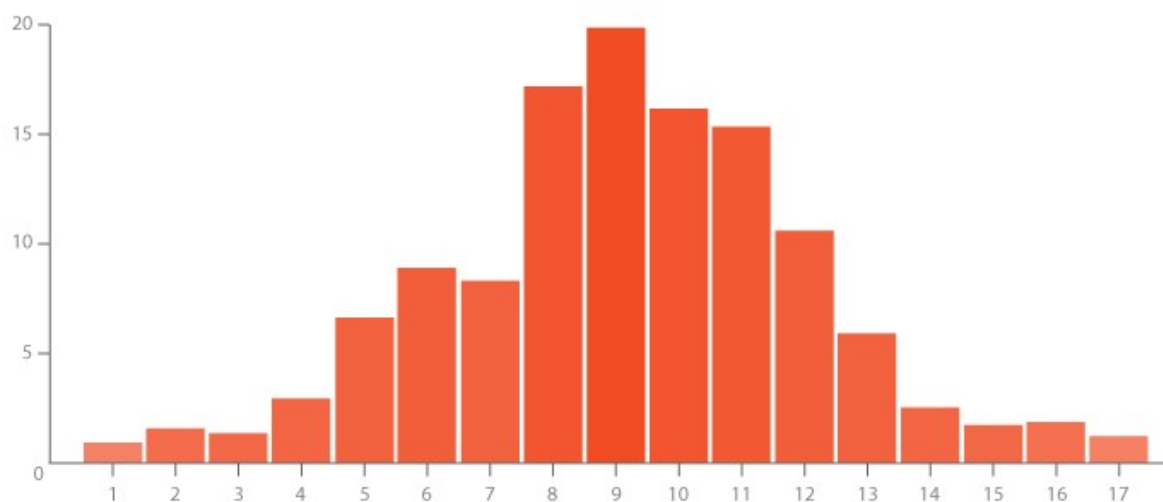
## Anatomy



### Correlation Strength:



# Histogram

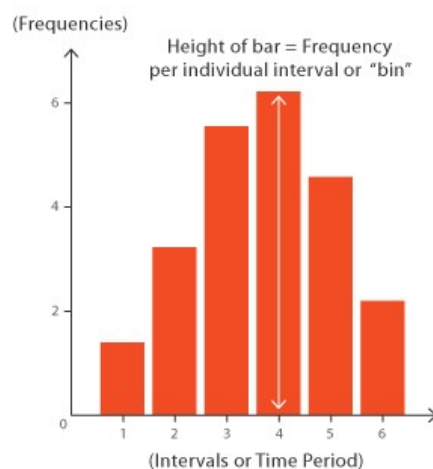


## Description

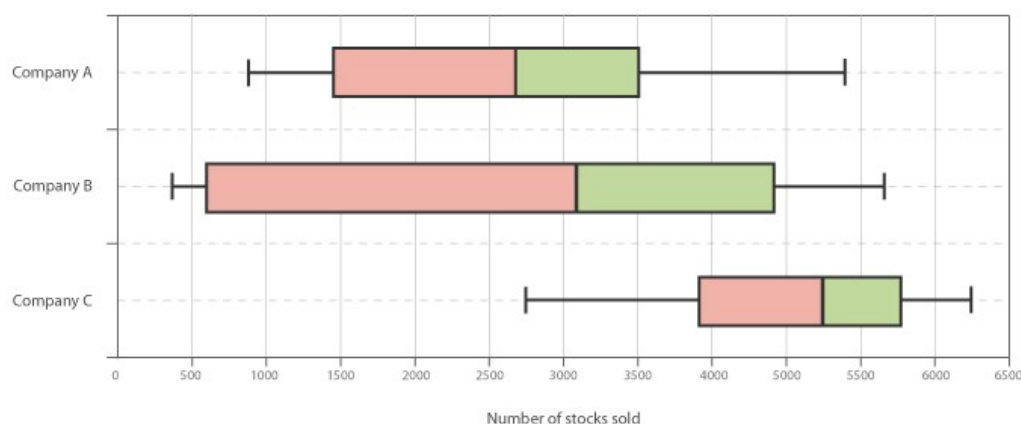
A Histogram visualises the distribution of data over a continuous interval or certain time period. Each bar in a histogram represents the tabulated frequency at each interval/bin.

Histograms help give an estimate as to where values are concentrated, what the extremes are and whether there are any gaps or unusual values. They are also useful for giving a rough view of the probability distribution.

## Anatomy



# Box and Whisker Plot



## Description

A Box and Whisker Plot (or Box Plot) is a convenient way of visually displaying the data distribution through their quartiles.

The lines extending parallel from the boxes are known as the “whiskers”, which are used to indicate variability outside the upper and lower quartiles. Outliers are sometimes plotted as individual dots that are in-line with whiskers. Box Plots can be drawn either vertically or horizontally.

Although Box Plots may seem primitive in comparison to a [Histogram](#) or [Density Plot](#), they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets.

Here are the types of observations one can make from viewing a Box Plot:

- What the key values are, such as: the average, median 25th percentile etc.

- If there are any outliers and what their values are.

- Is the data symmetrical.

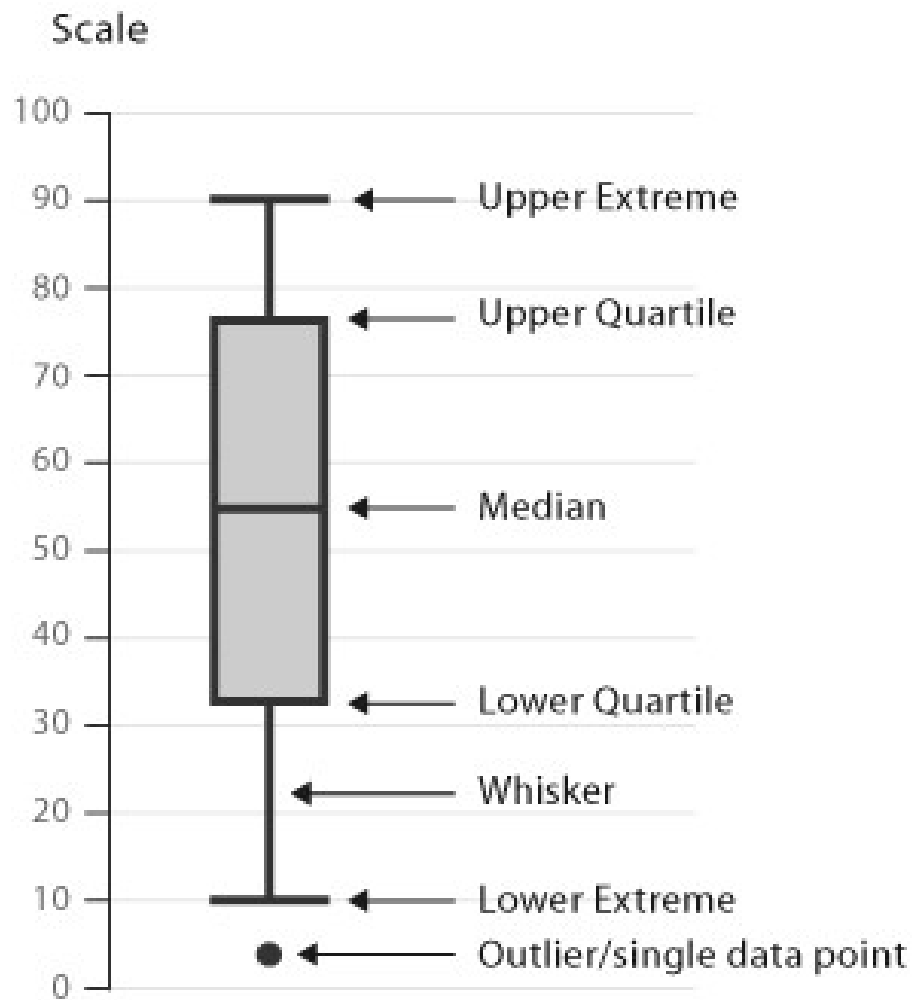
- How tightly is the data grouped.

- If the data is skewed and if so, in what direction.

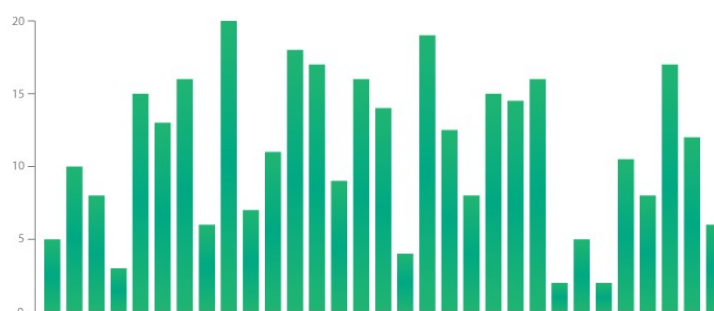
Two of the most commonly used variation of Box Plot are: variable-width Box Plots and notched Box Plots.

# Box and Whisker Plot

## Anatomy



## Bar Chart



## Description

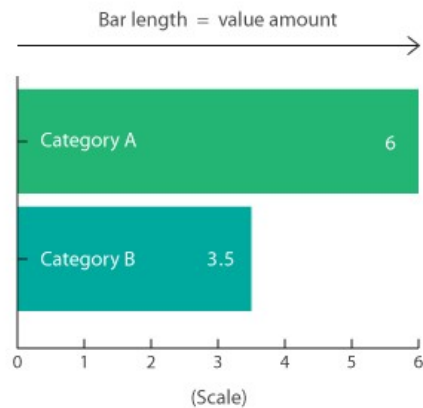
As known as *Bar Graph* or *Column Graph*.

The classic Bar Chart uses either horizontal or vertical bars (column chart) to show discrete, numerical comparisons across categories. One axis of the chart shows the specific categories being compared and the other axis represents a discrete value scale.

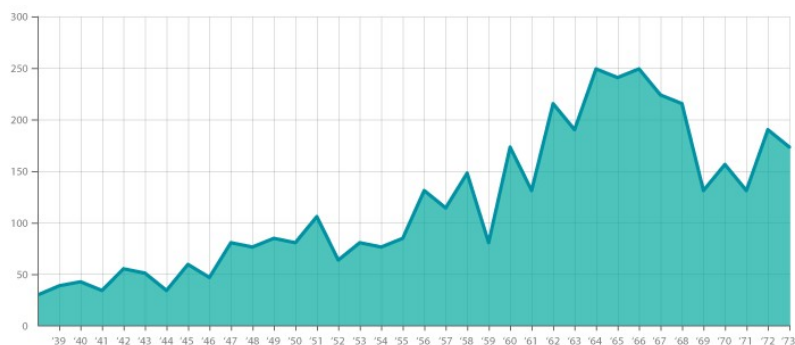
Bars Charts are distinguished from [Histograms](#), as they do not display continuous developments over an interval. Bar Chart's discrete data is categorical data and therefore answers the question of "how many?" in each category.

One major flaw with Bar Charts is that labelling becomes problematic when there are a large number of bars.

## Anatomy



## Area Graph



## Description

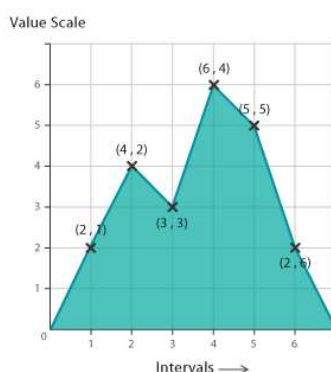
Area Graphs are [Line Graphs](#) but with the area below the line filled in with a certain colour or texture.

Area Graphs are drawn by first plotting data points on a Cartesian coordinate grid, joining a line between the points and finally filling in the space below the completed line.

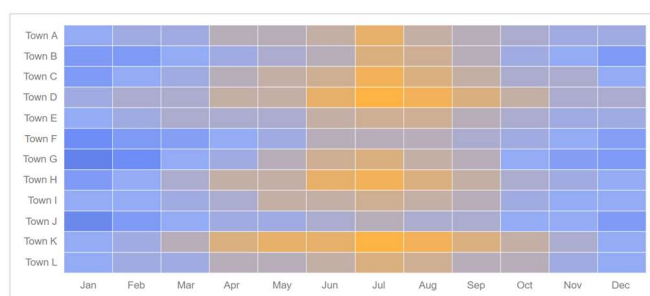
Like Line Graphs, Area Graphs are used to display the development of quantitative values over an interval or time period. They are most commonly used to show trends, rather than convey specific values.

Two popular variations of Area Graphs are: grouped and [Stacked Area Graphs](#). Grouped Area Graphs start from the same zero axis, while Stacked Area Graphs have each data series start from the point left by the previous data series.

## Anatomy



## Heatmap (Matrix)



## Description

Heatmaps visualise data through variations in colouring. When applied to a tabular format, Heatmaps are useful for cross-examining multivariate data, through placing variables in the rows and columns and colouring the cells within the table. Heatmaps are good for showing variance across multiple variables, revealing any patterns, displaying whether any variables are similar to each other, and for detecting if any correlations exist in-between them.

Typically, all the rows are one category (labels displayed on the left or right side) and all the columns are another category (labels displayed on the top or bottom). The individual rows and columns are divided into the subcategories, which all match up with each other in a matrix. The cells contained within the table either contain colour-coded categorical data or numerical data, that is based on a colour scale. The data contained within a cell is based on the relationship between the two variables in the connecting row and column.

A legend is required alongside a Heatmap in order for it to be successfully read. Categorical data is colour-coded, while numerical data requires a colour scale that blends from one colour to another, in order to represent the difference in high and low values. A selection of solid colours can be used to represent multiple value ranges (0-10, 11-20, 21-30, etc) or you can use a gradient scale for a single range (for example 0 - 100) by blending two or more colours together.

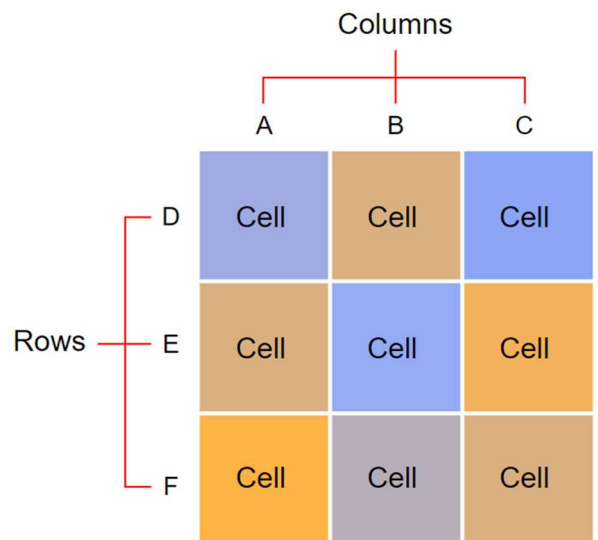
Because of their reliance on colour to communicate values, Heatmaps are a chart better suited to displaying a more generalised view of numerical data, as it's harder to accurately tell the differences between colour shades and to extract specific data points from (unless of course, you include the raw data in the cells).

Heatmaps can also be used to show the changes in data over time if one of the rows or columns are set to time intervals. An example of this would be to use a Heatmap to compare the temperature changes across the year in multiple cities, to see where's the hottest or coldest places. So the rows could list the cities to compare, the columns contain each month and the cells would contain the temperature values.

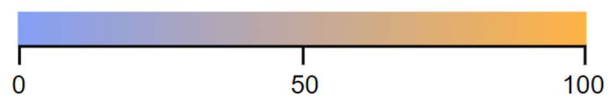
## Heatmap (Matrix)

### Anatomy

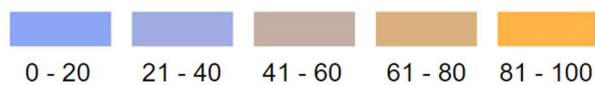
Heatmap using numerical data:



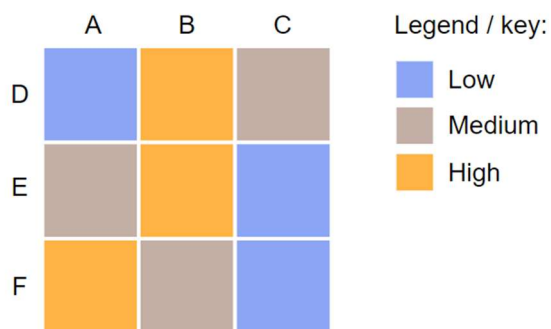
Value scale for determining cell colouring:



Alternative value scale broken into ranges:



Heatmap using categorical data:



## Treemap



## Description

Treemaps are an alternative way of visualising the hierarchical structure of a [Tree Diagram](#) while also displaying quantities for each category via area size. Each category is assigned a rectangle area with their subcategory rectangles nested inside of it.

When a quantity is assigned to a category, its area size is displayed in proportion to that quantity and to the other quantities within the same parent category in a part-to-whole relationship. Also, the area size of the parent category is the total of its subcategories. If no quantity is assigned to a subcategory, then its area is divided equally amongst the other subcategories within its parent category.

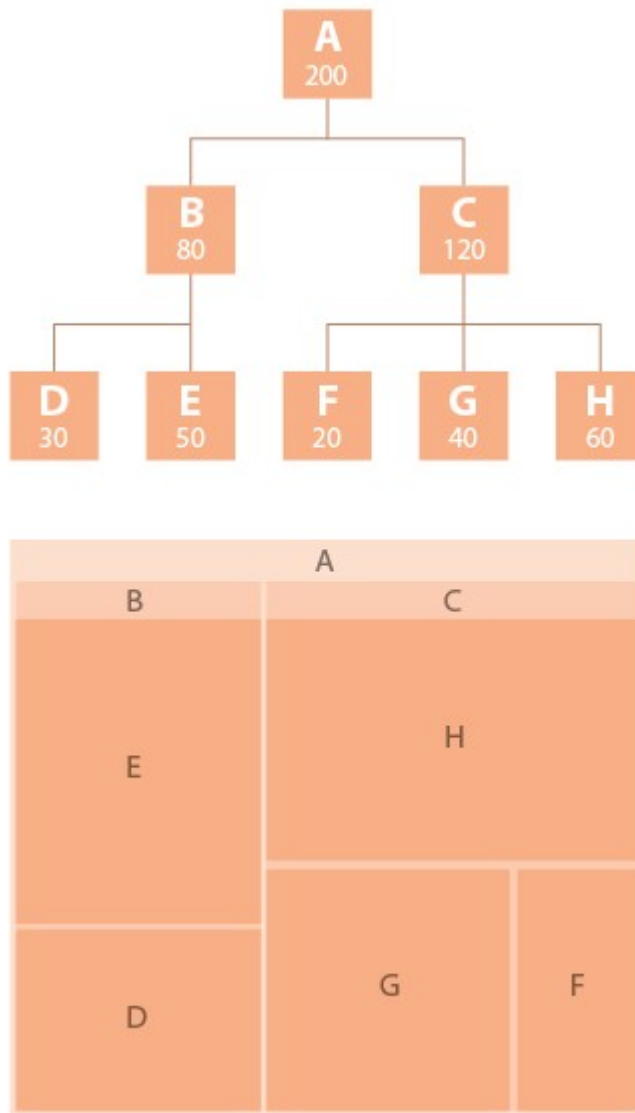
The way rectangles are divided and ordered into sub-rectangles is dependent on the tiling algorithm used. Many tiling algorithms have been developed, but the "squarified algorithm" which keeps each rectangle as square as possible is the one commonly used.

Ben Shneiderman originally developed Treemaps as a way of visualising a vast file directory on a computer, without taking up too much space on the screen. This makes Treemaps a more compact and space-efficient option for displaying hierarchies, that gives a quick overview of the structure. Treemaps are also great at comparing the proportions between categories via their area size.

The downside to a Treemap is that it doesn't show the hierarchal levels as clearly as other charts that visualise hierarchal data (such as a Tree Diagram or Sunburst Diagram).

## Treemap

## Anatomy



## Donut Chart



## Description

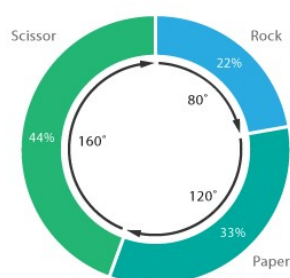
A donut chart is essentially a Pie Chart with an area of the centre cut out.

Pie Charts are sometimes criticised for focusing readers on the proportional areas of the slices to one another and to the chart as a whole. This makes it tricky to see the differences between slices, especially when you try to compare multiple Pie Charts together.

A Donut Chart somewhat remedies this problem by de-emphasizing the use of the area. Instead, readers focus more on reading the length of the arcs, rather than comparing the proportions between slices.

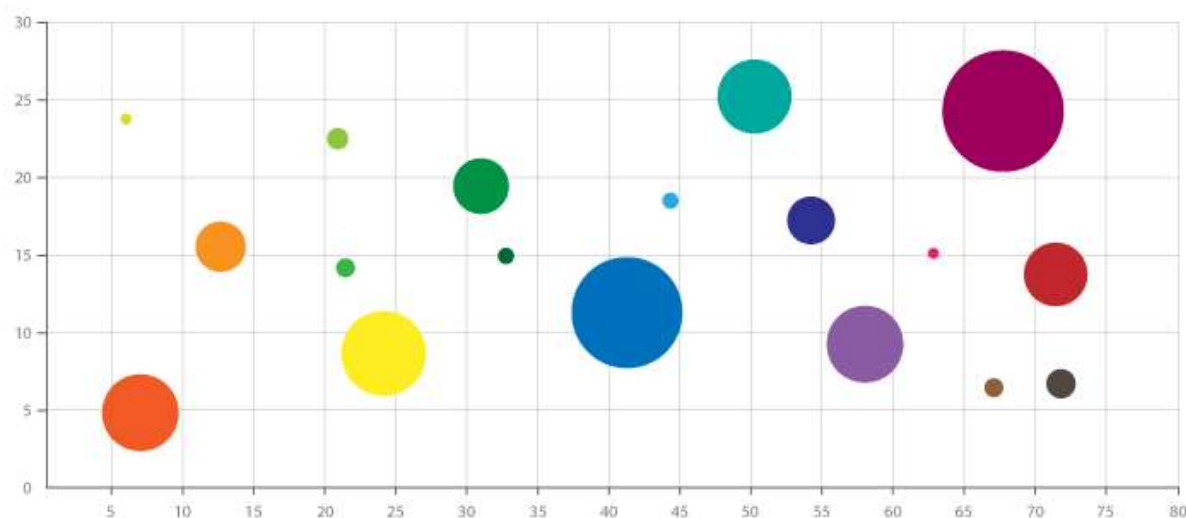
Also, Donut Charts are more space-efficient than Pie Charts because the blank space inside a Donut Chart can be used to display information inside it.

## Anatomy



Data			
Rock	Paper	Scissor	TOTAL
2	3	4	9
To calculate percentages			
$2/9 = 22\%$	$3/9 = 33\%$	$4/9 = 44\%$	100%
Degrees for each "donut slice"			
$(2/9) \times 360 = 80^\circ$	$(3/9) \times 360 = 120^\circ$	$(4/9) \times 360 = 160^\circ$	360°

## Bubble Chart



## Description

A Bubble Chart is a multi-variable graph that is a cross between a [Scatterplot](#) and a [Proportional Area Chart](#).

Like a Scatterplot, Bubble Charts use a Cartesian coordinate system to plot points along a grid where the X and Y axis are separate variables. However, unlike a Scatterplot, each point is assigned a label or category (either displayed alongside or on a legend). Each plotted point then represents a third variable by the area of its circle. Colours can also be used to distinguish between categories or used to represent an additional data variable. Time can be shown either by having it as a variable on one of the axis or by animating the data variables changing over time.

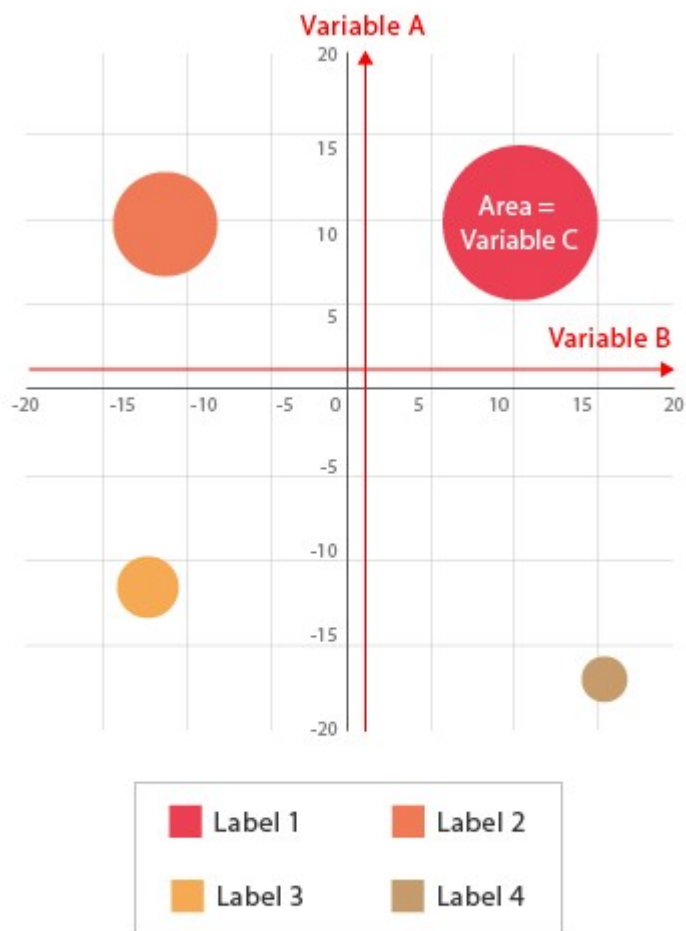
Bubble Charts are typically used to [compare](#) and show the [relationships](#) between categorised circles, by the use of [positioning](#) and [proportions](#). The overall picture of Bubble Charts can be used to analyse for [patterns/correlations](#).

Too many bubbles can make the chart hard to read, so Bubble Charts have a limited data size capacity. This can be somewhat remedied by interactivity: clicking or hovering over bubbles to display hidden information, having an option to reorganise or filter out grouped categories.

Like with Proportional Area Charts, the sizes of the circles need to be drawn based on the circle's area, not its radius or diameter. Not only will the size of the circles change exponentially, but this will lead to misinterpretations by the human visual system.

## Bubble Chart

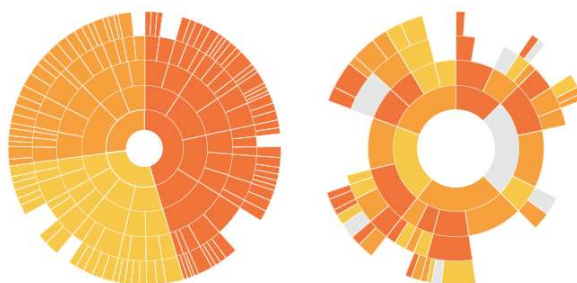
## Anatomy



$$\text{Circle Area} = \pi \times \text{Radius}^2$$

$$\text{Circle Diameter} = (\text{SQRT}(\text{Area} / \pi)) \times 2$$

## Sunburst Diagram



## Description

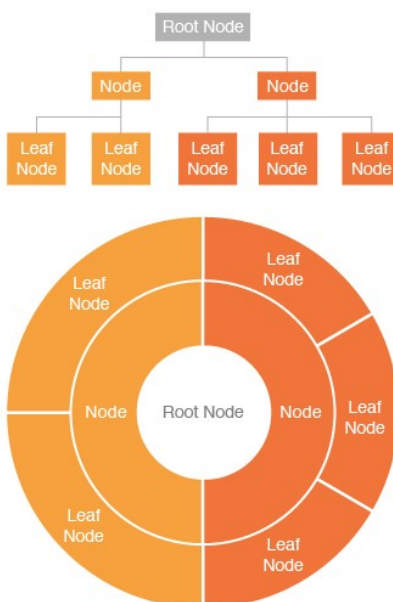
As known as a *Sunburst Chart*, *Ring Chart*, *Multi-level Pie Chart*, *Belt Chart*, *Radial Treemap*.

This type of visualisation shows hierarchy through a series of rings, that are sliced for each category node. Each ring corresponds to a level in the hierarchy, with the central circle representing the root node and the hierarchy moving outwards from it.

Rings are sliced up and divided based on their hierarchical relationship to the parent slice. The angle of each slice is either divided equally under its parent node or can be made proportional to a value.

Colour can be used to highlight hierarchal groupings or specific categories.

## Anatomy



# Integrating Custom Visualizations in Teradata AppCenter

**teradata.**

## Built-in Teradata AppCenter Visualizations

teradata.

2

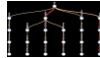
- Bar chart



- Chord



- Decision tree



- Hierarchical clustering tree



- Line chart



- Pie chart



- Sankey



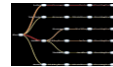
- Sigma



- Statistics Line



- Tree



- Wordcloud



- Wordbubbles wordcloud



© Teradata 2020

Bar chart compares at least one set of data points using x and y axes. Clustered bar charts are effective for showing multiple data sets. Stacked bar charts help with assessing proportion across a data set.

Chord charts interrelationships between data in a matrix. Ex. affinity of products bought together.

Decision tree is a type of flow diagram with endpoint values to assist in decision-making. Ex loan approval decision

Hierarchical Clustering visualizes data that has a containment relationship. Ex. USA → CA → Sacramento, San Francisco

Line chart are used to show trends with x and y axes

Pie chart illustrate break down in an individual dimension and represent proportion across categories

Sankey is a specific type of flow diagram in which the width of the arrows is shown proportionally to the flow quantity.

Sigma visualization is appropriate for depicting data networks and how they relate to each other

Statistics line is a type of line chart that support cfilter visualization formats

Tree is a type of flow diagram that supports npath visualization formats

Wordcloud is a visual representation of text data that can show proportion between

values

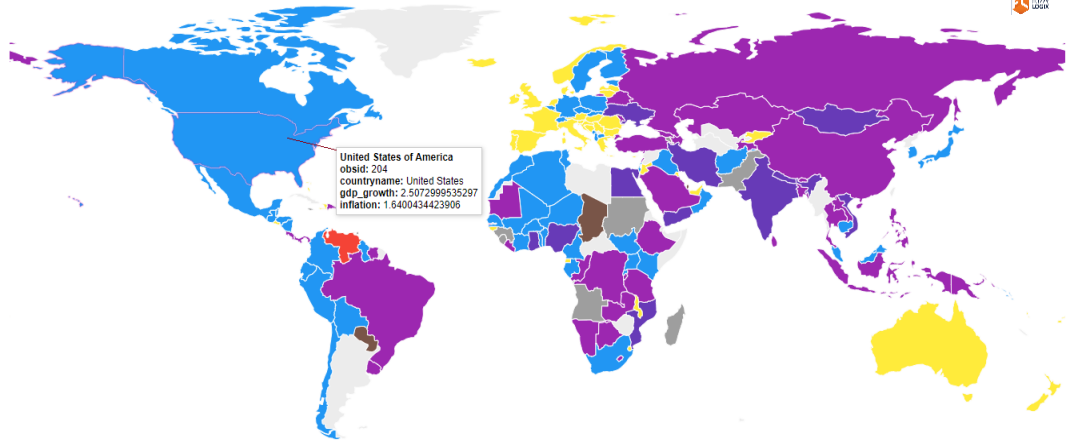
Wordbubbles Wordcloud is a variation of the wordcloud that uses bubble size to represent proportion

## End goal : Visualize GDP growth and Inflation of countries

teradata.



3



## Visualization quotes

teradata.

4

“

The purpose of visualization is insight, not pictures.

– Ben Shneiderman  
Computer scientist, a Distinguished University Professor in the University of Maryland Department of Computer Science.

“

Numbers have an important story to tell. They rely on you to give them a clear and convincing voice.

– Stephen Few  
IT innovator, consultant, and educator with over 30 years of experience in the fields of business intelligence and information design

“

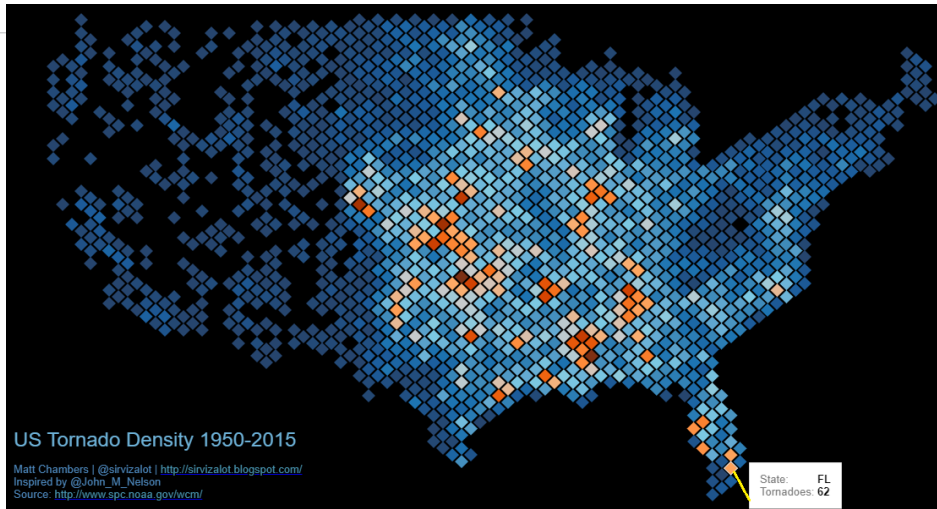
The greatest value of a picture is when it forces us to notice what we never expected to see.

– John w Tukey,  
Mathematician best known for development of the Fast Fourier Transform algorithm and box plot.

## Examples of Custom Visualizations\* : Hexbin map

teradata.

5



© Teradata 2020

\* Not available as a built-in AppCenter

1. Custom visualizations are visualizations that are freely downloadable from the web.
2. Each visualization is a self-contained HTML page written in Java Script, D3.js, HTML5 and CSS.
3. Every visualization expects data to be transformed to the format that the visualization expects.

A hexbin map, organizes areas around some specific geometries , in this case squares, and these are called bins. The idea is to visualize density as color intensity within each bin.

## Examples of Custom Visualizations\*: Chloropleth map

teradata.

6



© Teradata 2020

\* Not available as a built-in AppCenter

Choropleth maps use color to highlight differences between areas of map. Values depicted are categorical- red or blue. But there could be other choropleth maps with numerical values

## Examples of Custom Visualizations\* : Stacked Area Chart teradata.

7



Google

© Teradata 2020

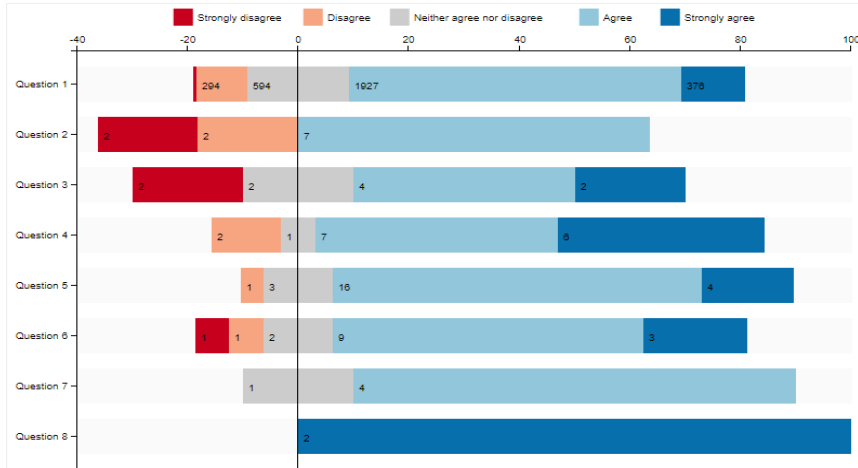
\* Not available as a built-in AppCenter

Music Timeline of a variety of music genres waxing and waning in popularity until 2010

## Examples of Custom Visualizations\*:Diverging stacked barchart

teradata.

8



© Teradata 2020

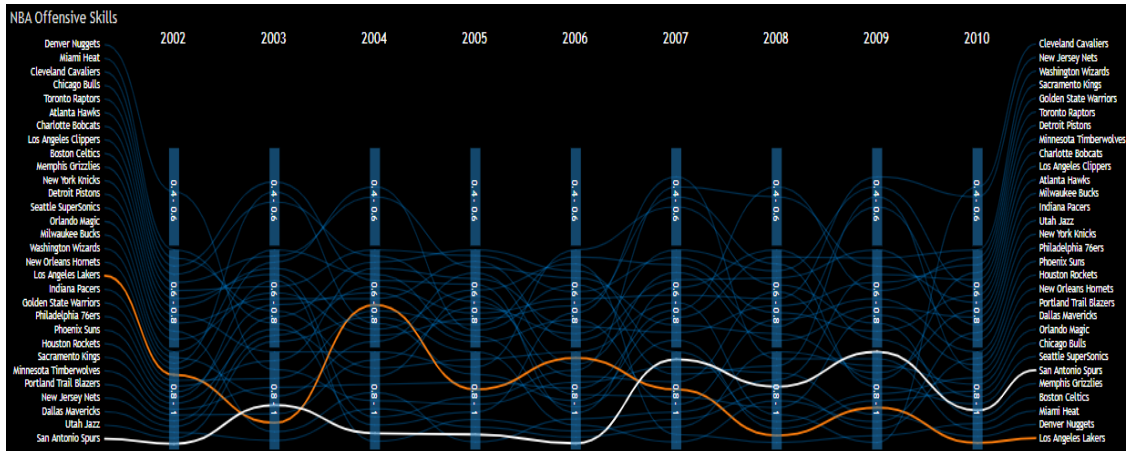
\* Not available as a built-in AppCenter

We create a diverging stacked bar chart to plot a 5 point Likert scale. There a lots of ways to plot a Likert scale but, a diverging stacked bar chart is the best

## Examples of Custom Visualizations\*: temporal line chart

teradata.

9



@Leto Peel et. al. ICDM 2015 paper "Predicting Sports Scoring Dynamics with Restoration and Anti-persistence"

© Teradata 2020

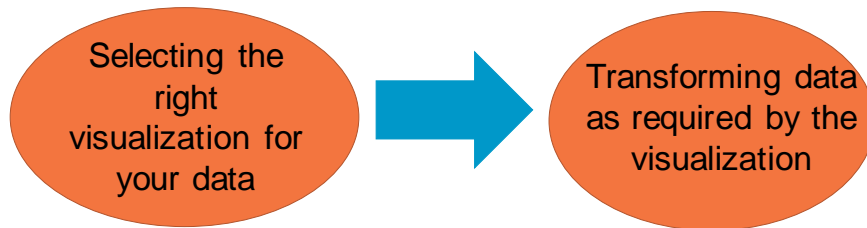
\* Not available as a built-in AppCenter

Inferred skills for each season of NBA (2002-2010) from Leto Peel et. al. ICDM 2015 paper "Predicting Sports Scoring Dynamics with Restoration and Anti-persistence". Teams that are the overall winners of more than one season are highlighted. Similar map with code is discussed at <https://bl.ocks.org/mbostock/3709000>

## The challenge of integrating a Custom Visualization with Teradata AppCenter lies in:

teradata.

10



© Teradata 2020

Where a treemap is required, a bar chart won't work.

For example, a grouped horizontal bar chart requires data to be a dictionary of dictionaries. Any other structure will not work.

## Steps to integrate a Custom Visualization

Select and  
download your  
visualization

Pull data from  
table in json  
format using  
query service

Package and  
upload the  
updated  
visualization

STEP 1

STEP 2

STEP 3

STEP 4

STEP 5

STEP 6

Identify input  
table to source  
data to  
visualization

Update  
visualization to  
accept data from  
json

Run the app

## Steps 1 and 2 :

teradata.

12

1. Select and download your custom visualization.
  - Downloaded the visualization to display GDP growth and Inflation rate of countries
2. Identify input table to source data to visualization.
  - Sampled rows from input table :

Obsid	CountryName	CountryCode	Cluster_id	GDP growth rate	Inflation Rate
204	United States	USA	5	2.5072999535297	1.6400434423906
69	United Kingdom	GBR	5	1.6597541606755	3.2857142857149
65	France	FRA	5	1.7247755947887	1.529639382277
51	Germany	DEU	5	4.0124659135177	1.1038085608358
89	India	IND	2	10.259963064554	11.992296918768
93	Iceland	ISL	5	-4.0987774858702	5.3965047672074

© Teradata 2020

`distinct(countrycode) = 175`

`distinct(cluster_id) = 7 (1,2,3,4,5,7)`

`select cluster_id,count(*) from medkmeansoutput group by cluster_id order by cluster_id;`

`1,1`

`2,11`

`3,39`

`4,61 // US is in this cluster`

`5,48`

`6,5`

`7,10`

## Step 3: Pull data from table using query service

teradata.

The screenshot shows the Teradata AppCenter interface for a script named 'country-map'. The left sidebar contains navigation options: Develop, Script Overview, Script Code, Description, Execution History, Script Results, Logs, Permissions, and Settings. The main content area is divided into sections: 'Setup Guide' (Required and optional setup steps), 'Script Code' (highlighted with a red box), 'Script User Options', 'Description (Optional)', 'Permissions (Optional)', and 'Latest Script Results'. The 'Script Code' field contains the following SQL code:

```
begin;
--name=query1
SELECT * FROM HEDMeansOutput;
end;
```

A red arrow points to the 'Script Code' field. The 'Script Info' section on the right shows script metadata and general settings, including 'country-map', 'Color blue-700, text volume\_mute', '600 MB Memory', 'Q1 CPU', and 'user1'.

© Teradata 2020

“--name=query1 ” helps associate the query with the visualization

...run and browse the json returned by query service **teradata.**

≡

teradata

APPCENTER

14

Job Results

01dc70d1-7620-4...

Oct 17, 2019, 10:57:57 PM

application/json

Result data set 1

Data table

JSON

Array[175]

> 0: Object

> 1: Object

> 2: Object

> 3: Object

> 4: Object

> 5: Object

> 6: Object

> 7: Object

> 8: Object

166: Object

obsid: 98

countryname: "Japan"

countrycode: "JPN"

cluster\_id: 5

gdp\_growth: 4.6521118555914

inflation: -0.71978158351906

167: Object

obsid: 204

countryname: "United States"

countrycode: "USA"

cluster\_id: 5

gdp\_growth: 2.5072999535297

inflation: 1.6400434423906

> 168: Object

> 169: Object

> 170: Object

© Teradata 2020

## Step 4: Update Visualization to accept data from json

teradata.

15

Visualization code updated to accept json

```
var m=[];
var dataObj = {}
var cdata = dataObj["results"][0]["data"];
var metadata = Object.keys(cdata[0]);
for(var k=0; k<cdata.length; k++)
{
    m.push(Object.values(cdata[k]));
}
process_data(m, metadata);
```

json returned by query service

```
JSON
  queueDuration : 2
  queryDuration : 103
  results
    0
      resultSet : true
      columns
        data
          0
            1
              2
                166
                  obsid : 95
                  countryname : "Japan"
                  countrycode : "JPN"
                  cluster_id : 5
                  gdp_growth : 4.6521110555914
                  inflation : -0.71978158351906
                167
                  obsid : 204
                  countryname : "United States"
                  countrycode : "USA"
                  cluster_id : 5
                  gdp_growth : 2.5072999535297
                  inflation : 1.6400434423906
                168
                169
                170
                171
                172
                173
                174
              rowCount : 175
```

© Teradata 2020

Run query without the "--name=query1" from AppCenter UI and the json composed by the query service on data that the query service pulled from source table will display.

## Step 5: Package and upload the updated Visualization

teradata.

16

The screenshot shows the Teradata AppCenter interface for the 'country-map' application. The left sidebar contains navigation options: Develop, Script Overview, Script Code, Description, Execution History, Script Results, Logs, Permissions, and Settings. The main content area is titled 'country-map Settings' and includes a 'Visualizations' section. This section has a dropdown menu set to 'Custom (Upload)' and a 'select visualization options' link. Below this, there are instructions for bundling a file to upload, including steps for installing Node.js, Polymer Bundler, and running the bundling process. A red box highlights these instructions, and a red arrow points to the 'BROWSE' button at the bottom right of the instructions.

© Teradata 2020

polymer-bundler is a library for packaging project assets for production to minimize network round-trips. Web pages that use multiple HTML Imports, external scripts, and stylesheets to load dependencies may end up making lots of network round-trips. In many cases, this can lead to long initial load times and unnecessary bandwidth usage. The polymer-bundler tool follows HTML Imports, external script and stylesheet references, inlining these external assets into "bundles", to be used in production.

## Step 6: Run the app

teradata.

17

The screenshot displays the Teradata AppCenter interface for a script named 'country-map'. The main panel shows script details, including a 'Script info' section with metadata (version 0.1, CPU 0.1, memory 600 MB) and a 'Script Code' section with SQL code. A yellow box highlights the 'RUN' button in the top right of the script details panel. To the right, a 'Run New Job' dialog is open, showing settings for the job. The 'Select System' dropdown is set to 'teradata-system', and the 'Database' is set to 'tda\_admin'. The 'Parameters' section is empty. The 'Schedule' section is set to 'Every' with a red arrow pointing to the 'Title and description' field. The 'RUN NOW' button is highlighted in green.

Script Details

country-map  
Script

RUN FAVORITE DEVELOP

Script info  
Information about this Script

user1  
Created

9/30/19, 5:34 PM  
Updated

user1  
Last Update By

sql script  
Type

Script info

0.1  
CPU

600 MB  
Memory

Latest Script Results

Description  
learn more about this script

Script Code

```
begin;  
DROP TABLE IF EXISTS MCDONALD;  
end;  
  
begin;  
CREATE TABLE MCDONALD  
(  
  FIELD INTEGER,  
  FIELD INTEGER  
)
```

Run New Job  
country-map

New job will be created and executed using all configured settings and default values

Settings

Select System  
Choose which system you'd like to query

System \*  
teradata-system

Database  
tda\_admin

Parameters  
Enter the required params to run this app

Schedule  
Set the job to run automatically at certain times

Every

Title and description  
Name and description for this job

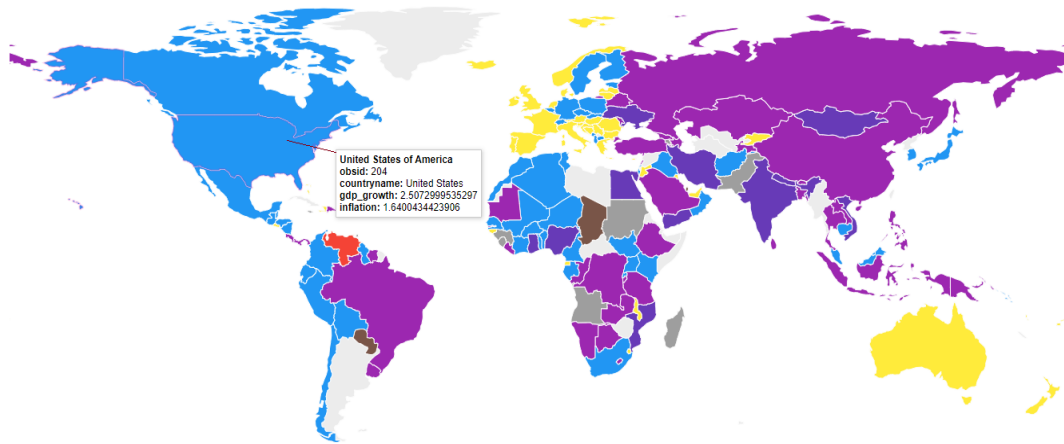
RUN NOW CANCEL

© Teradata 2020

## The Custom Visualization is ready

teradata.

18



## References

- Teradata AppCenter User Guide
- Readings in Information Visualization: Using Vision to Think  
Jock D. Mackinlay and Stuart Card
- A Tour through the Visualization Zoo  
Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky, Stanford University  
July/August 2019 issue of AcmQueue
- Show Me the Numbers: Designing Tables and Graphs to Enlighten  
Stephen Few
- Information Dashboard Design: The Effective Visual Communication of Data  
Stephen Few
- <http://perceptualedge.com>

## References

- Great Academic paper on D3  
D3 Data-Driven Documents  
Authors: Michael Bostock, Vadim Ogievetsky, Jeffrey Heer  
IEEE Transactions on Visualization and Computer Graphics archive  
Volume 17 Issue 12, December 2011
- IEEE Transactions on Visualization and Computer Graphics
- Effectiveness of Animation in Trend Visualization  
George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko

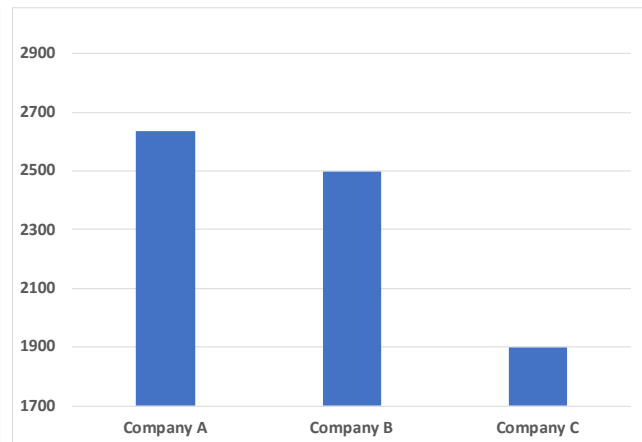
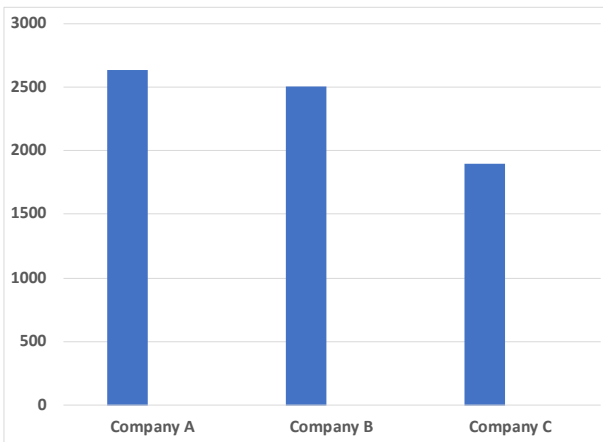
## Misleading Graphs

### Bar plots

A truncated graph is a graph (usually seen in bar plots) where the y-axis labels doesn't start at zero. Truncated graphs are typically used to highlight differences between categories, but these can lead to misinformation and tend to overexaggerate the differences.

Since the scale changes, these graphs would seem to represent the data differently, thus people generally overestimate and often incorrectly interpret the actual differences.

The following two graphs below demonstrate this, the chart on the left shows a bar plot starting at 0, while the one on the right starts at a higher number (1700). While they use exactly the same data, the chart on the right, shows a huge gap between, Company C, which might be interpreted as Company C performing way below as compared to Company A and Company B, at about 75% less. But in actuality, Company C is performing only about 30% less than that of Company A / B.



## The Boxplot and Its Pitfalls

**Source:** <https://www.data-to-viz.com/caveat/boxplot.html#:~:text=A%20boxplot%20can%20summarize%20the,higher%20value%20than%20the%20others.>

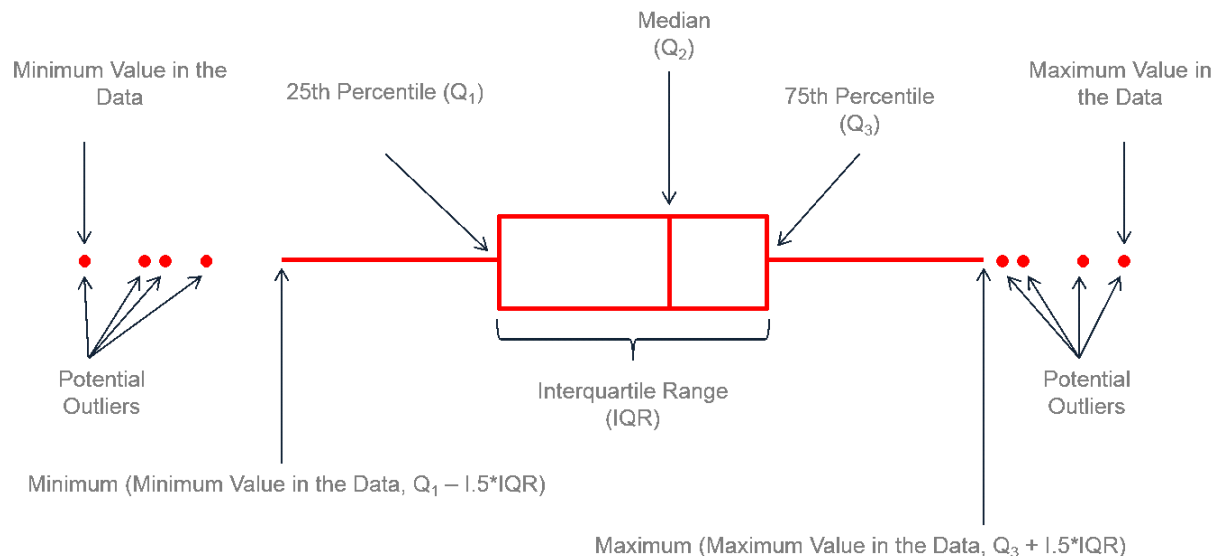
### Code

A boxplot gives a nice summary of one or more numeric variables. A boxplot is composed of several elements:

The line that divides the box into 2 parts represents the **median** of the data. If the median is 10, it means that there are the same number of data points below and above 10.

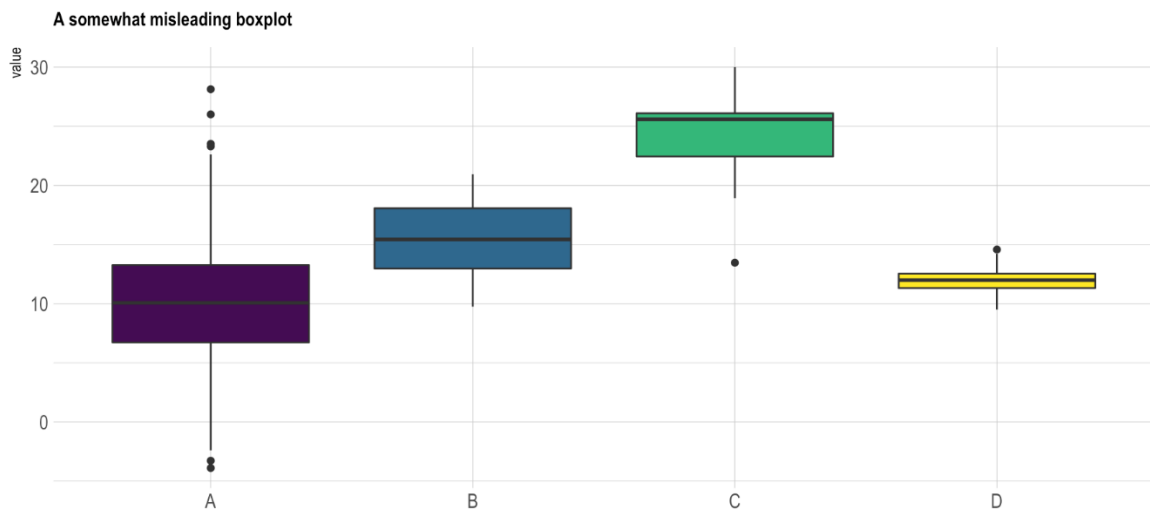
- The ends of the box shows the upper (Q3) and lower (Q1) **quartiles**. If the third quartile is 15, it means that 75% of the observation are lower than 15.
- The difference between Quartiles 1 and 3 is called the **interquartile range (IQR)**
- The extreme line shows  $Q3 + 1.5 \times IQR$  to  $Q1 - 1.5 \times IQR$  (the highest and lowest value excluding outliers).
- Dots (or other markers) beyond the extreme line shows potential outliers.

Here is a diagram showing the boxplot anatomy:



A boxplot can summarize the distribution of a numeric variable for several groups. The problem is that summarizing also means losing information, and that can be a pitfall. If we consider the boxplot below, it is easy to conclude that group C has a higher value than the others. However, we cannot see the underlying distribution of dots in each group or their number of observations.

## Code

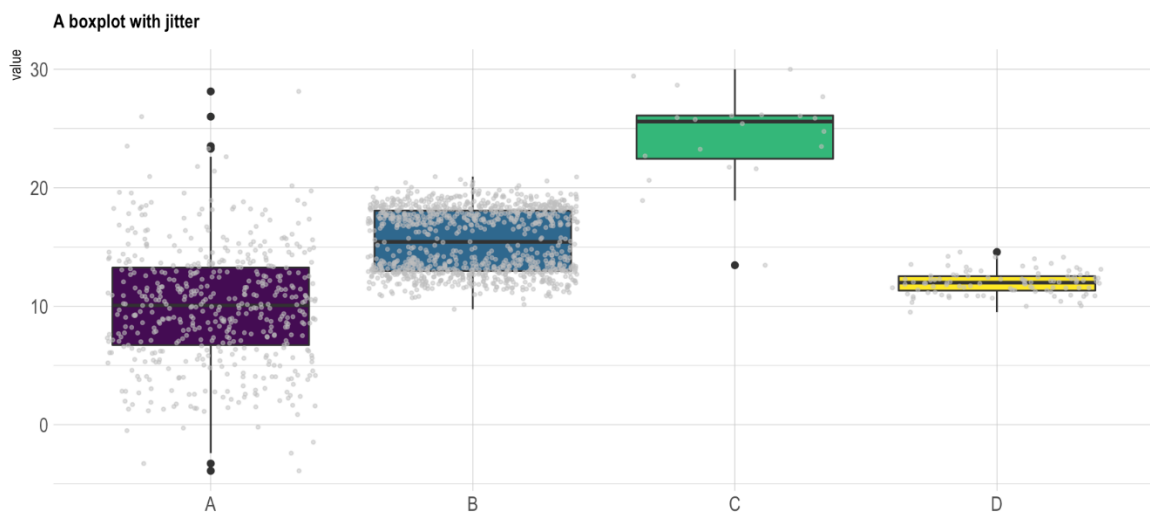


Let's see what happens when the boxplot is improved using additional elements.

## Adding jitter

If the amount of data you are working with is not too large, adding jitter on top of your boxplot can make the graphic more insightful.

## Code



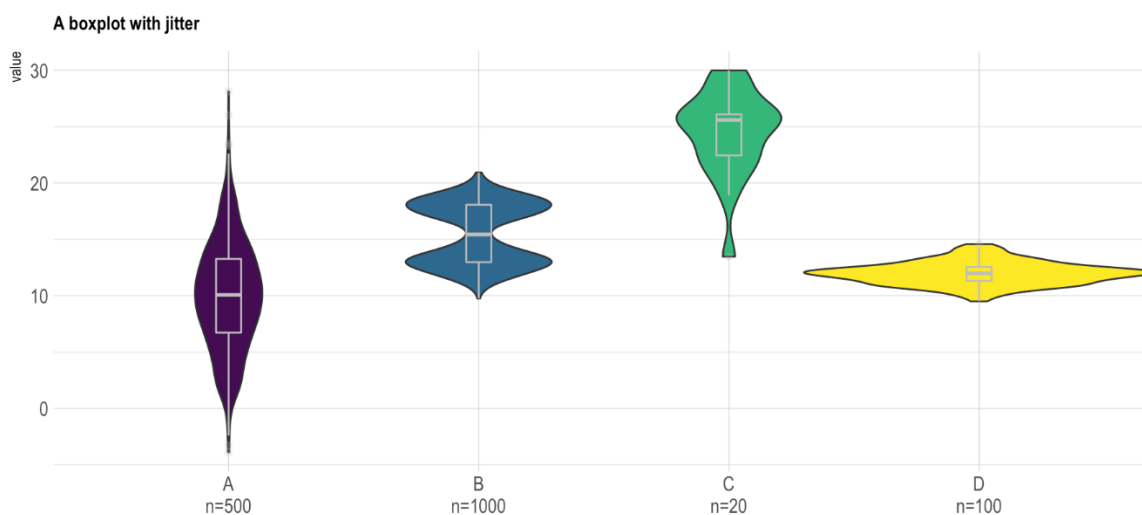
Here some new patterns appear clearly. Group C has a small sample size compared to the other groups. This is definitely something you want to find out before saying that group C has a higher value than the others. Moreover, it looks like group B has a bimodal distribution: dots are distributed in 2 groups: around  $y=18$  and  $y=13$ .

Switching to violin plot

---

If you have a large sample size, using jitter is not an option anymore since dots will overlap, making the figure uninterpretable. A alternative is the [violin plot](#), which describes the distribution of the data for each group:

CODE



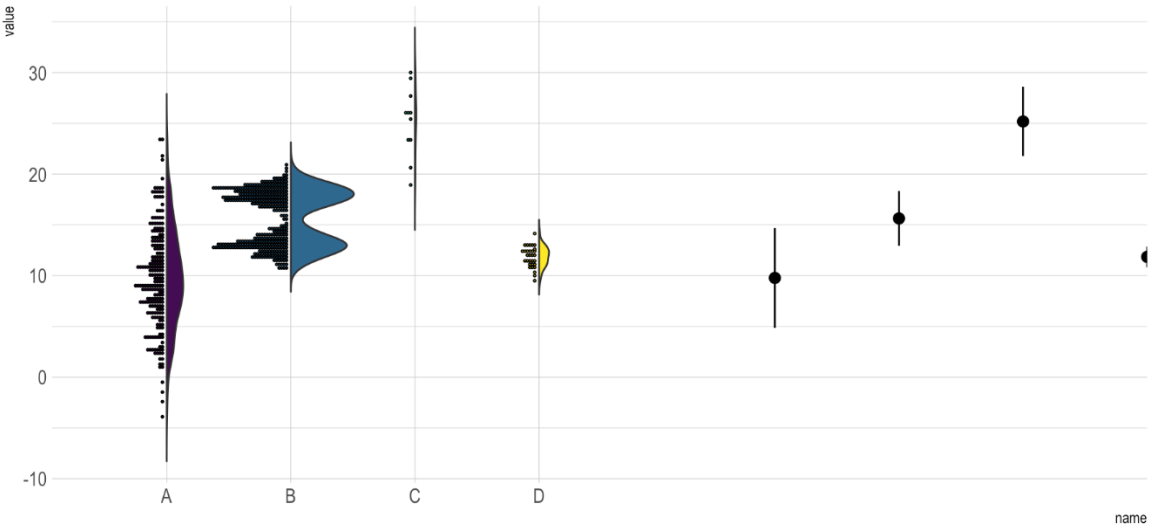
Here it is very clear that the groups have different distributions. The bimodal distribution of group B becomes obvious. Violin plots are a powerful way to display information—they are probably under-utilized compared to boxplots.

Adding the sample size

---

On the previous chart, the sample size of each group is indicated on the x-axis, below the group name. This is a good practice and shows that group C is under-represented. However, it is sometimes better to show the data points themselves. Thus, a good alternative is a half violin plot showing the raw data. This uses code coming from [jbburant](#) and [David Robinson](#).

CODE



Source: [https://www.data-to-viz.com/caveat/calculation\\_error.html](https://www.data-to-viz.com/caveat/calculation_error.html)

## CALCULATION ERRORS

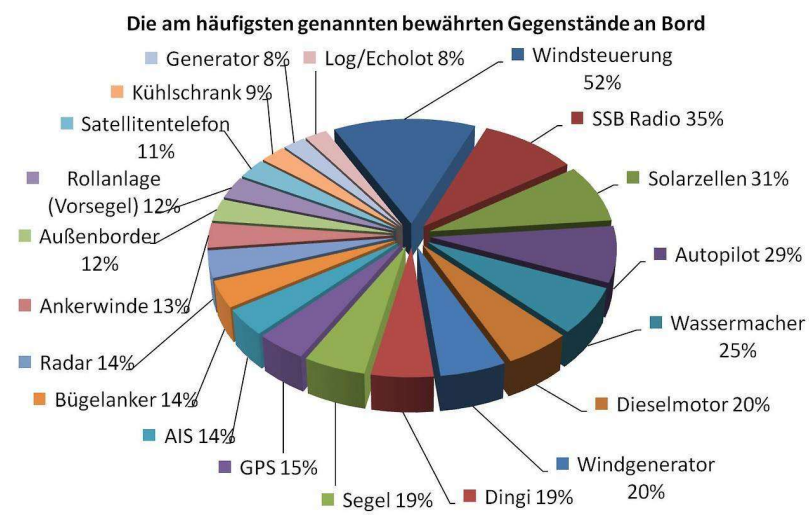
A collection of common [dataviz caveats](#) by [Data-to-Viz.com](#)



This is probably the most obvious pitfall example of the collection, but is probably the most frequent one as well. Number inconsistencies on a graphic make it completely useless.

## Percentages don't add up to 100%

For example, when displaying percentages on a pie chart, double-check that the percents sum to 100%:



Source: [WTF Visualizations](#)

Here, also note that using an exploded 3D pie chart is probably the worst way to convey information ever invented. ([Read more about it](#))

## Values don't match visuals

In the following example, the 45% annotation is linked with the biggest part of the donut chart; clearly something is incorrect.



Source: [WTF Visualizations](#)

Source: <https://www.data-to-viz.com/caveat/pie.html>

# THE ISSUE WITH PIE CHART

*A collection of common [dataviz caveats](#) by [Data-to-Viz.com](#)*



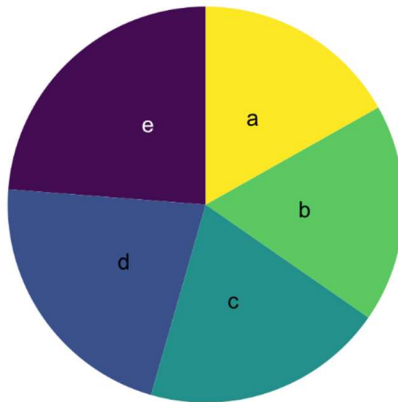
CODE

## Bad by definition

A pie chart is a circle divided into sectors that each represent a proportion of the whole. It is often used to show percentage, where the sum of the sectors equals 100%.

The problem is that humans are pretty bad at reading angles. In the adjacent pie chart, try to figure out which group is the biggest one and try to order them by value. You will probably struggle to do so and this is why pie charts must be avoided.

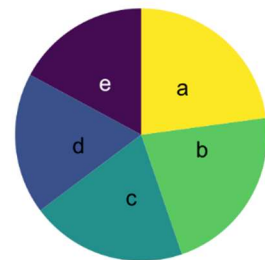
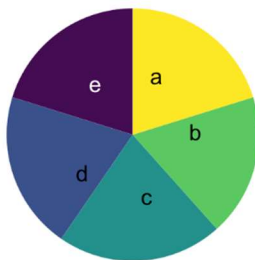
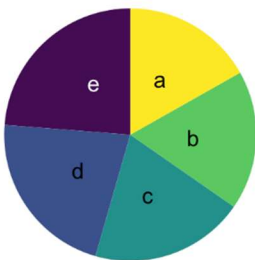
CODE



---

If you're still not convinced, let's try to compare several pie plots. Once more, try to understand which group has the highest value in these 3 graphics. Also, try to figure out what is the evolution of the value among groups.

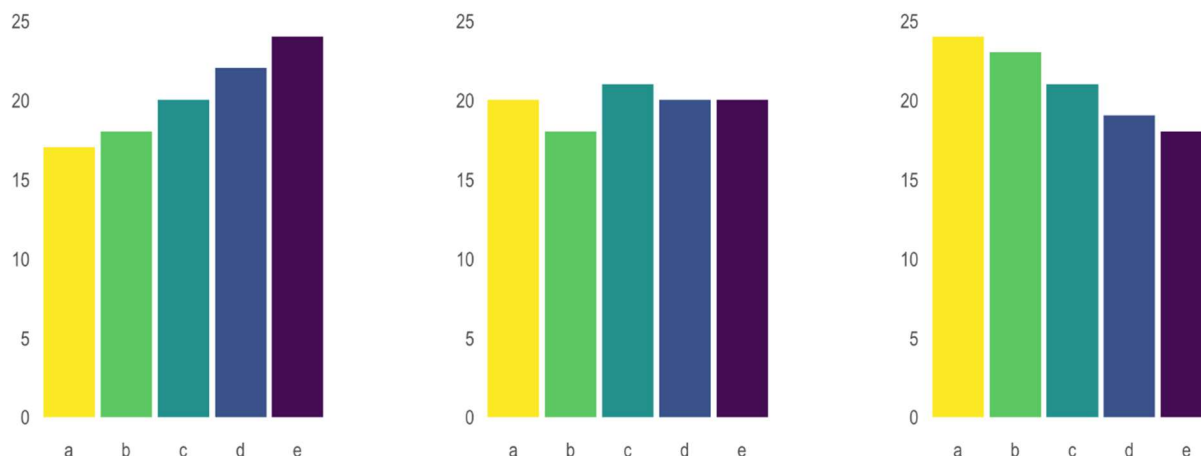
CODE



---

Now, let's represent exactly the same data using a [barplot](#):

CODE



As you can see on this barplot, there is a heavy difference between the three pie plots with a hidden pattern that you definitely don't want to miss when you tell your story.

## And often made even worse

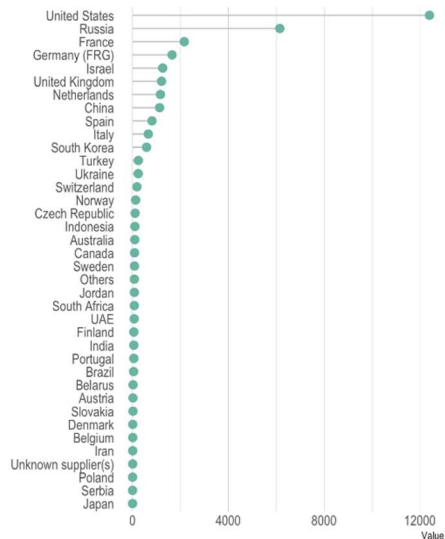
Even if pie charts are bad by definition, it is still possible to make them even worse by adding other bad features:

- 3d
- legend aside
- percentages that do not sum to 100
- too many items
- exploded pie charts

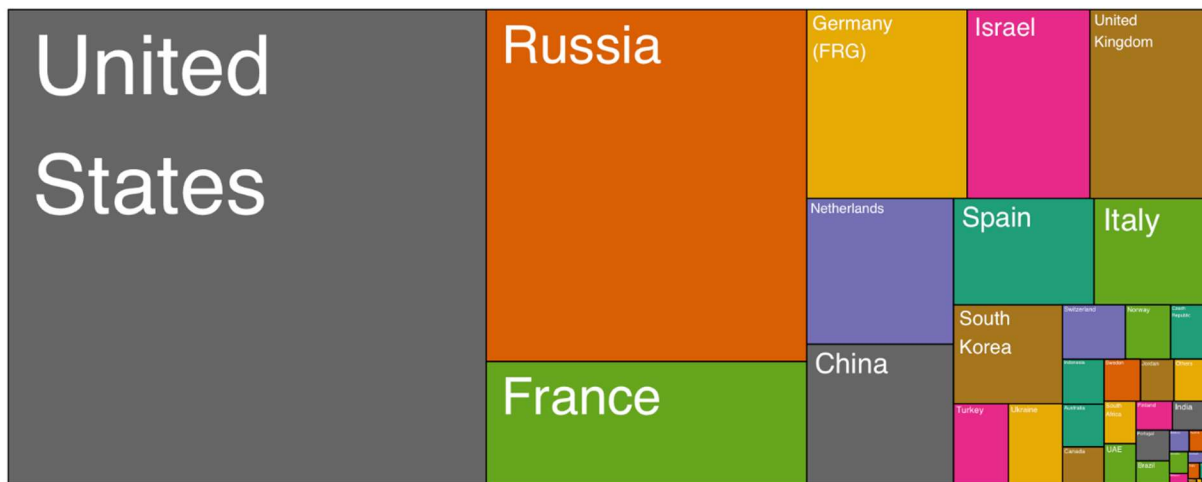
## Alternatives

The [barplot](#) is the best alternative to pie plots. If you have many values to display, you can also consider a [lollipop](#) plot that is a bit more elegant in my opinion. Here is an example based on the [amount of weapons sold](#) by a few countries in the world:

## CODE



Another possibility would be to create a treemap if your goal is to describe what the whole is composed of.

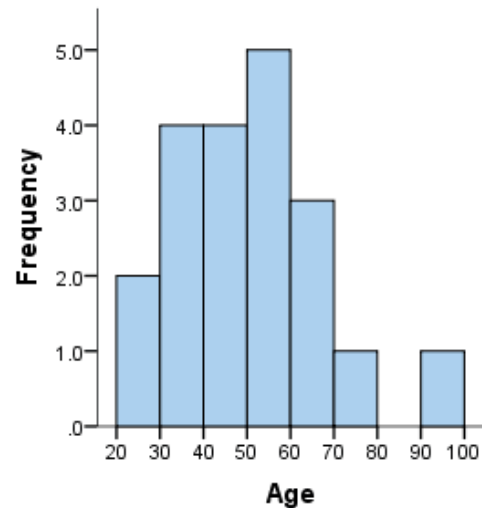


# Histograms

**Source:** <https://statistics.laerd.com/statistical-guides/understanding-histograms.php>

## What is a histogram?

A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. An example of a histogram, and the raw data it was constructed from, is shown below:



36	25	38	46	55	68	72	55	36	38
67	45	22	48	91	46	52	61	58	55

## How do you construct a histogram from a continuous variable?

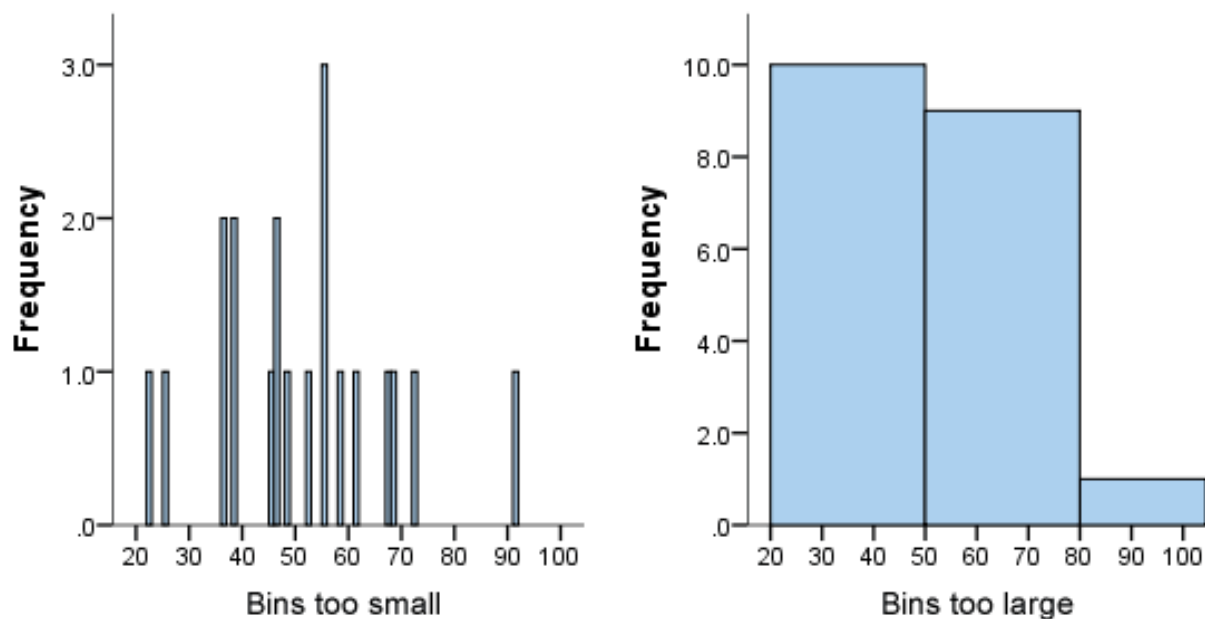
To construct a histogram from a continuous variable you first need to split the data into intervals, called **bins**. In the example above, **age** has been split into bins, with each bin representing a 10-year period starting at 20 years. Each bin contains the number of occurrences of scores in the data set that are contained within that bin. For the above data set, the frequencies in each bin have been tabulated along with the scores that contributed to the frequency in each bin (see below):

Bin	Frequency	Scores Included in Bin
20-30	2	25,22
30-40	4	36,38,36,38
40-50	4	46,45,48,46
50-60	5	55,55,52,58,55
60-70	3	68,67,61
70-80	1	72
80-90	0	-
90-100	1	91

Notice that, unlike a bar chart, there are no "gaps" between the bars (although some bars might be "absent" reflecting no frequencies). This is because a histogram represents a continuous data set, and as such, there are no gaps in the data (although you will have to decide whether you round up or round down scores on the boundaries of bins).

### Choosing the correct bin width

There is no right or wrong answer as to how wide a bin should be, but there are rules of thumb. You need to make sure that the bins are not too small or too large. Consider the histogram we produced earlier (see above): the following histograms use the same data, but have either much smaller or larger bins, as shown below:



We can see from the histogram on the left that the bin width is too small because it shows too much individual data and does not allow the underlying pattern (frequency distribution) of the data to be easily seen. At the other end of the scale is the diagram on the right, where the bins are too large, and again, we are unable to find the underlying trend in the data.

### Histograms are based on area, not height of bars

In a histogram, it is the area of the bar that indicates the frequency of occurrences for each bin. This means that the height of the bar does not necessarily indicate how many occurrences of scores there were within each individual bin. It is the product of height multiplied by the width of the bin that indicates the frequency of occurrences within that bin. One of the reasons that the height of the bars is

---

often incorrectly assessed as indicating frequency and not the area of the bar is due to the fact that a lot of histograms often have equally spaced bars (bins), and under these circumstances, the height of the bin does reflect the frequency.

**What is the difference between a bar chart and a histogram?**

The major difference is that a histogram is only used to plot the frequency of score occurrences in a continuous data set that has been divided into classes, called bins. Bar charts, on the other hand, can be used for a great deal of other types of variables including ordinal and nominal data sets.

Source: <https://blog.datawrapper.de/dualaxis/>

# Why not to use two axes, and what to use instead

The case against dual axis charts



**TL;dr: We believe that charts with two different y-axes make it hard for most people to intuitively make right statements about two data series. We recommend two alternatives strongly: using two charts instead of one and using indexed charts.**

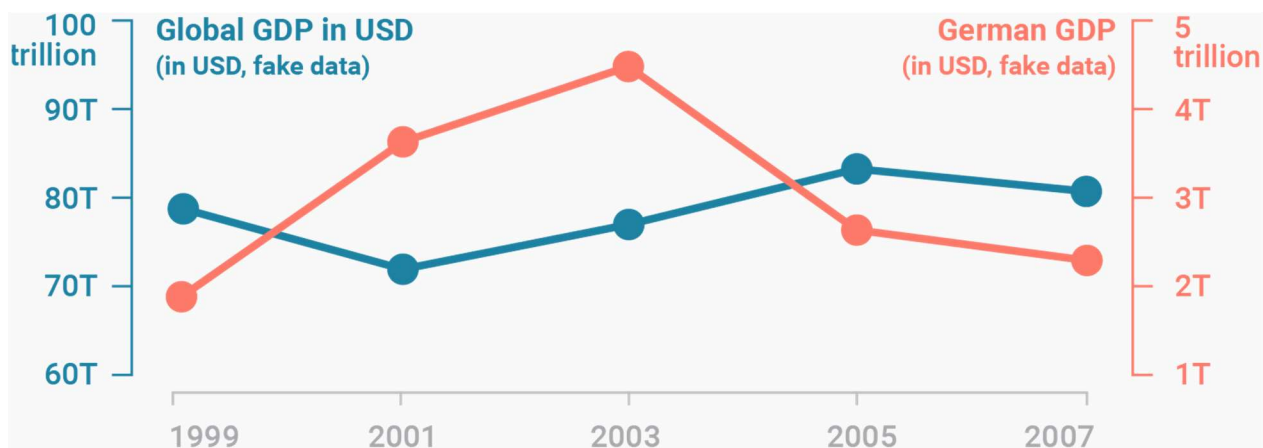
From time to time we get an email asking if it's possible in our data visualization tool [Datawrapper](#) to create charts with two different y-axes (also called double Y charts, dual axis charts, dual-scale data charts or superimposed charts). It is not – and we won't add it any time soon. We're sorry if that makes our user's life harder, but we agree with the many chart experts<sup>[1]</sup> who make cases against dual axis charts. We hope you'll hear us out.

We will first look at situations when people want to use dual axis charts, then we explain their problems, and afterward we'll look at four alternatives:

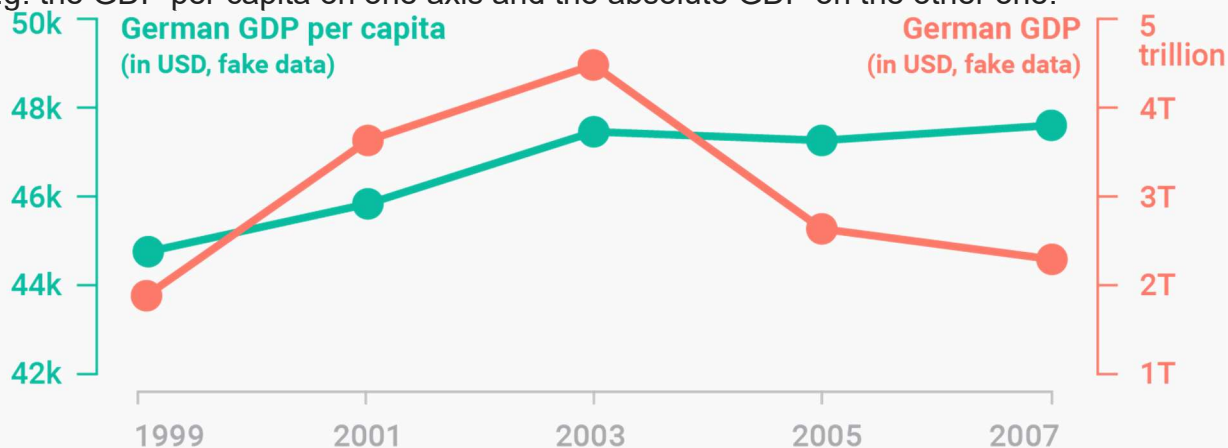
## Why people use dual axis charts

Why do people use dual axis charts? We looked around and found that most people used them to show...

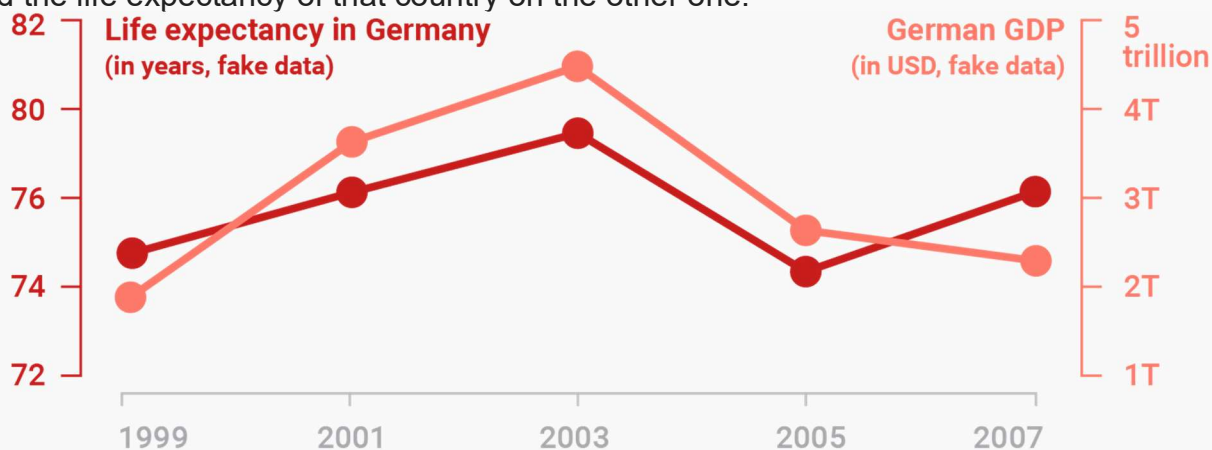
**1 ...two data series with the same measure, but different magnitudes**, e.g. the global GDP on one axis and the GDP of Germany on the other one:



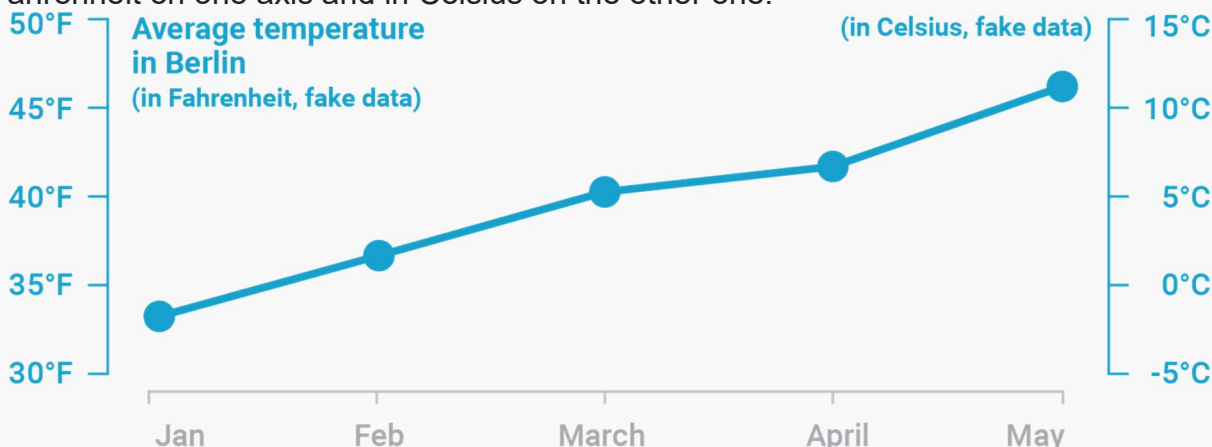
**2 ...two data series that show the relative and the absolute values of something, e.g. the GDP per capita on one axis and the absolute GDP on the other one:**



**3 ...two data series for totally different values, e.g. the GDP of a country on one axis and the life expectancy of that country on the other one:**



**4 ...one data series, but the y-axis shows different scales**, e.g. the values in Fahrenheit on one axis and in Celsius on the other one:



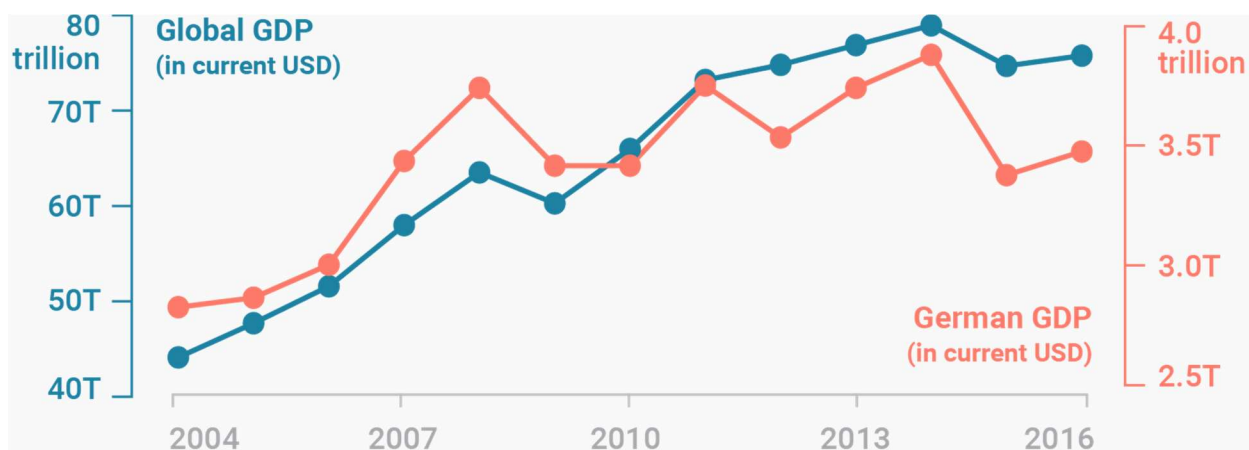
As you can see, dual axis charts are often used to show two different data series with a **different magnitude (=number range) and/or measure (GDP, life expectancy, etc)**. Often, their goal is to **compare two trends with each other**. Giving readers the possibility to do so makes a lot of sense – but there are some reasons why a dual axis chart is not the way to go. In fact, of these four use cases, we think that only the last dual axis chart can be used without any doubt, since it only uses the second Y-axis to show an alternative scale and not a second data series.

Let's have a look at the problems with dual axis charts before thinking about alternatives:

## The problems with dual axis charts

Here's the problem in a nutshell: **The scales of dual axis charts are arbitrary and can therefore (deliberately) mislead readers about the relationship between the two data series.**

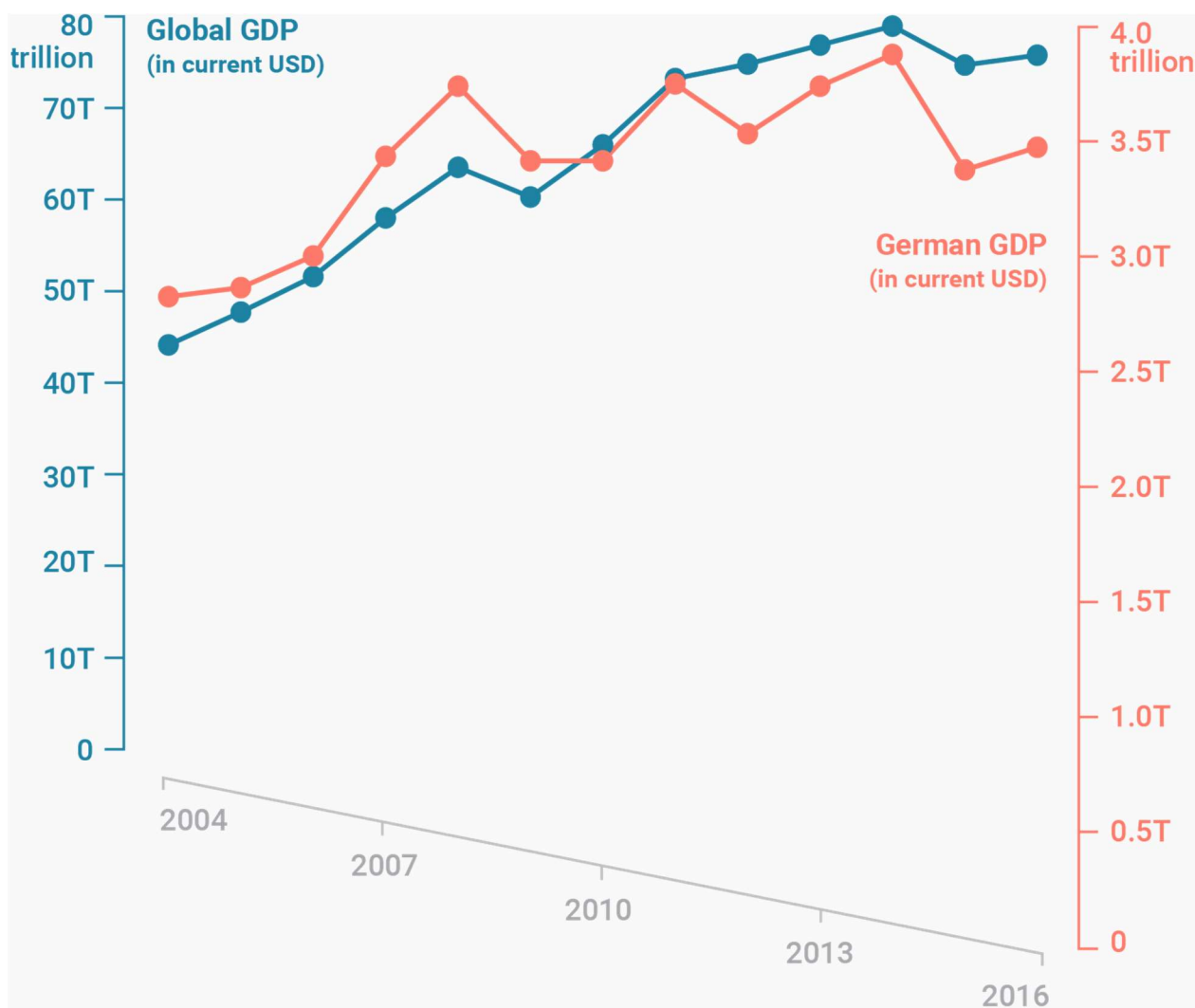
Let's use some real [Worldbank data](#) for the German GDP and the global GDP between 2004 and 2016 to explain that:



This chart has two different y-axes: The left axis shows the global GDP with a range from \$40 to \$80 trillion. The right axis shows the German GDP with a range between \$2.5 and \$4 trillion. The measure (US-Dollar) is the same, but we have a wildly different magnitude. A second axis sounds like a good solution – but there are three problems we have with them:

### Zero baselines at different heights can mislead

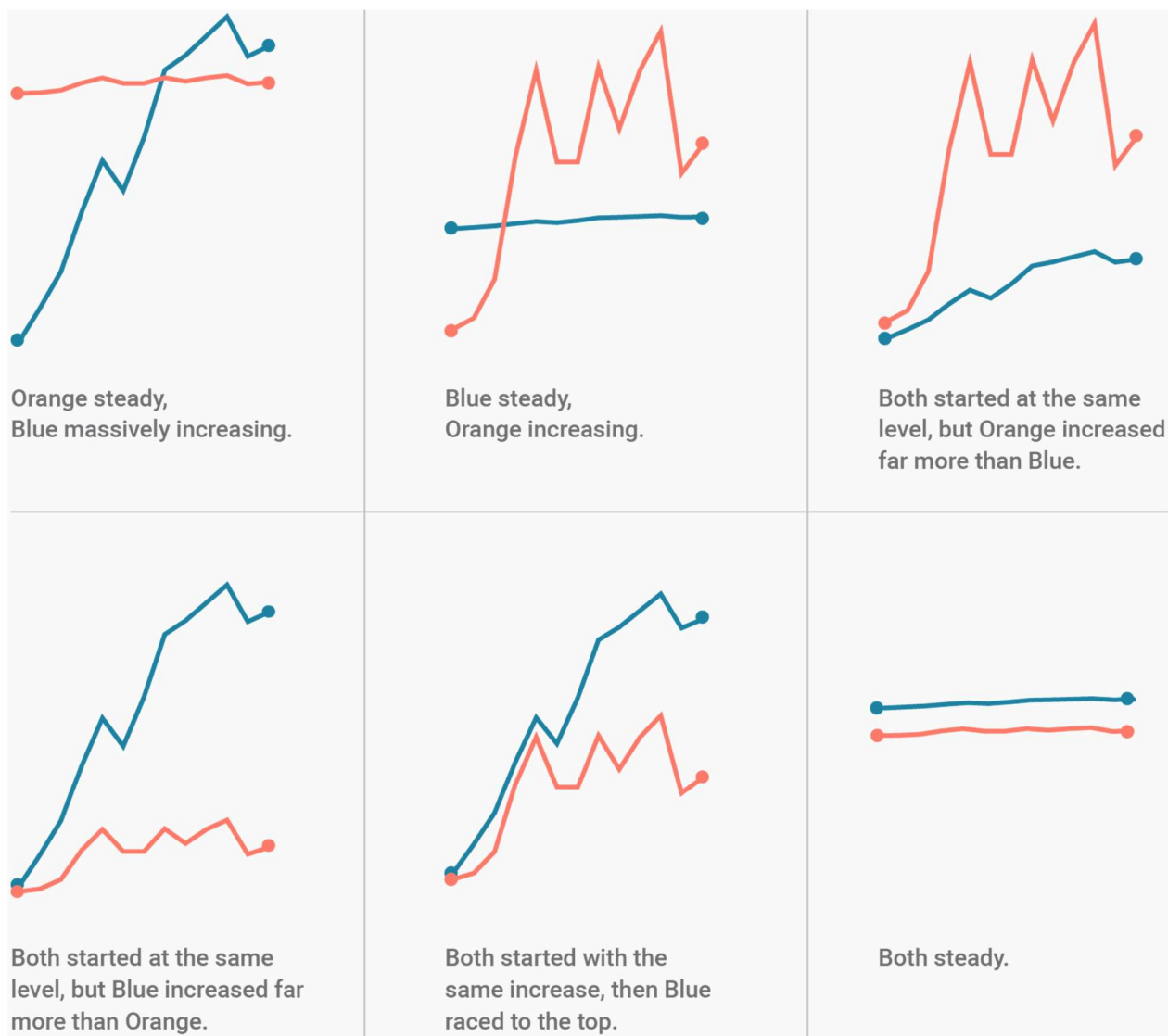
The proportions of the two scales are often different from each other in dual axis charts. If the left axis would go down to zero, the chart would be twice as long. If the right axis would go down to zero, the chart would be almost three times as long. This is how both axes look like when we extend them to zero:



So while **the chart looks like** the German GDP and the global GDP go up at roughly the same rate (at least until 2014), they don't. The global GDP increased by 80% until 2014; the GDP of Germany by 40%.

Most readers are used to line charts with just one scale. So when they see a line chart with two scales, their intuition goes into the normal “that’s how I read a line chart”-mode: “Oh, two lines, cool, same rate, interesting”. Readers actively need to remind themselves that these two lines have less of a relationship than they’re used to seeing in a line chart.

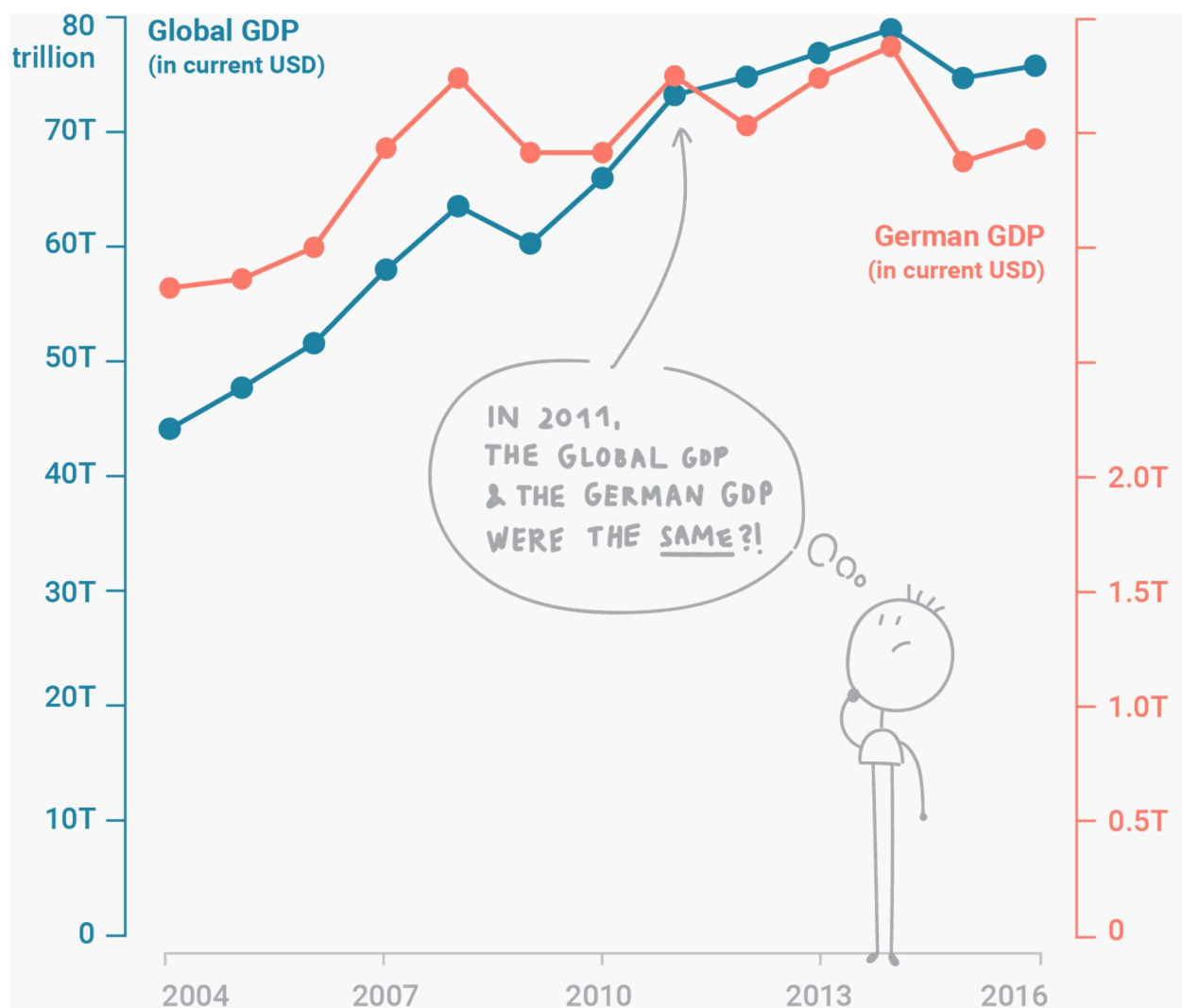
So how small is the relationship between these two lines? Let’s go crazy. Nothing really matters, right? **We can make all kinds of statements with our two data sets if we just tweak the scales a little bit:**



But that's a problem we can solve, isn't it? We can just set the zero baseline to the same height. Except:

### Even zero baselines at the same height can mislead

This is how the chart looks like with the same baseline. (Meaning, if we extended both y-axes to zero, they would have the same height.) In the best case, our readers will now think: "Seems like the global GDP increased more than the German GDP". Yes! Success! Except, in the worst case, our readers will think: "In the first years, the German GDP was higher than the global GDP. And then in 2011, the two GDP's were the same."



Why would anyone think that? Because humans have a tendency to set things in relation [if they're close-by](#), and this relationship becomes a huge part of the meaning they see in things. Data points and data series are not an exception to this rule. We automatically compare lines and points with each other; and it's hard to remember that different scales are involved. **If things look close-by on a chart, it's hard to constantly remember that actually, they are miles apart.**

### They're just hard to read

"Ha," you might say, "readers just need to look closer. I stared at this chart for a minute and I figured everything out." Well, good for you. **But most of our readers don't like to do math in their heads. (Which is ok: Our job is to do the math for them.)**

A [study from 2011](#) backs up that claim. Petra Isenberg, Anastasia Bezerianos, Pierre Dragicevic and Jean-Daniel Fekete showed 15 people four different charts that all showed values in different magnitudes, and observed how well these people could read

the charts. One of them was a chart with a dual axis, which the researchers call “superimposed chart”. That’s what they found out:

We found across the board that the superimposed chart performed poorly both in terms of accuracy and time. Participants’ feedback from the questionnaire was also clearly against the superimposed chart and it was ranked lowest by all but one participant. Participants called it very confusing and demanding too much concentration or reflection to decipher the non-monotonic and discontinuous nature of the two scales. – [A Study on Dual-Scale Data Charts](#)

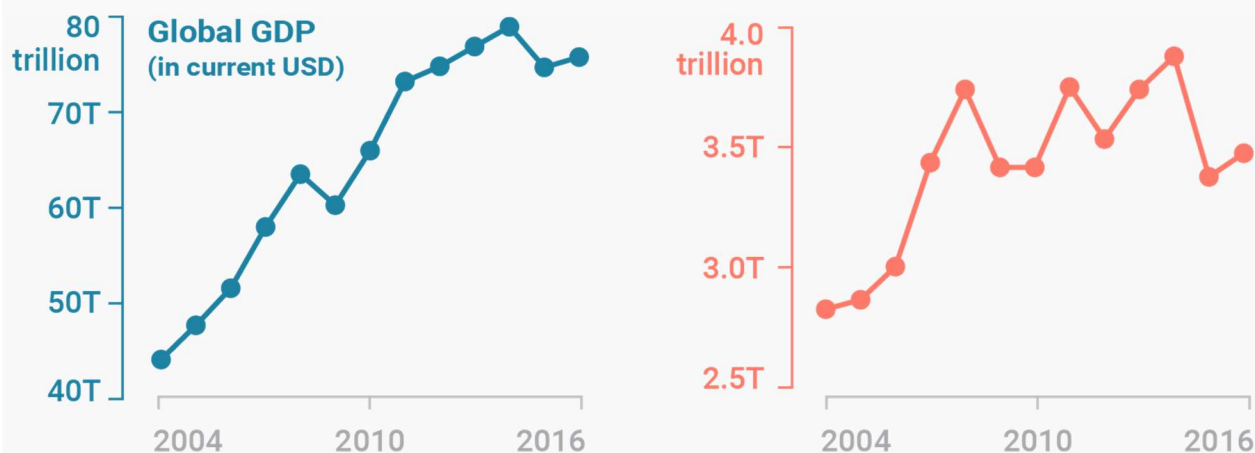
The researchers go on and recommend to avoid dual axis charts altogether. We agree. We tried to show here that the danger of dual axis charts is that they’re not intuitive. Chart designers have the freedom to manipulate axes as they wish, which can lead to first visual impressions which are way off what the data actually says.

## Alternatives

However, there’s hope! There are alternatives. Here we will present four of them: Creating two charts, indexed charts, labeling and connected scatterplots.

### Solution 1: Side-by-side charts

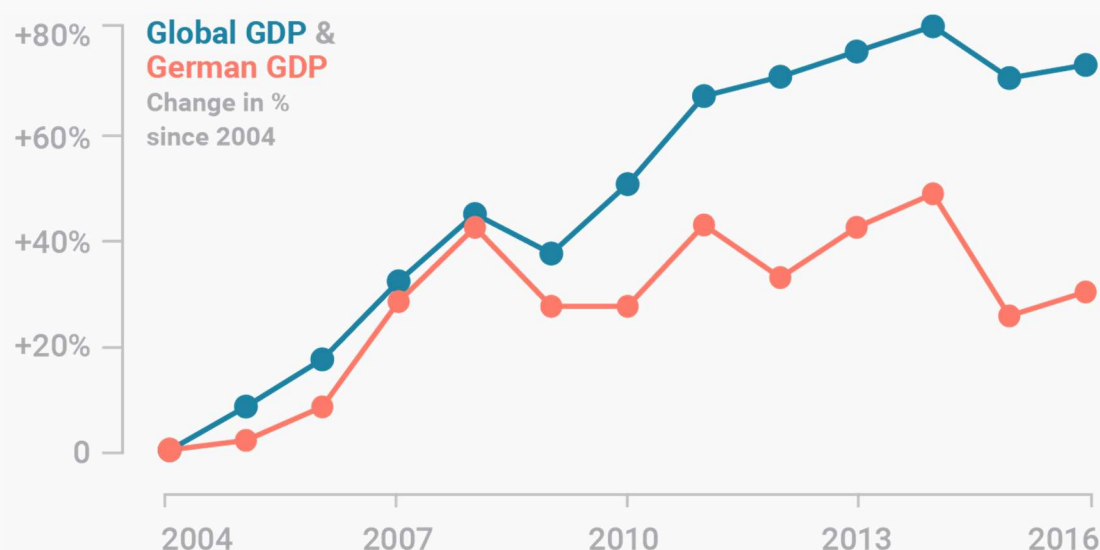
If the problem is that the two lines create meaning because they’re so close together, let’s separate them! The first solution is to create two different charts with our two data series, also called side-by-side-charts. The advantage is that – like with a dual axis chart – side-by-side charts **don’t care how much the numbers differ**: We can create two different axes for two different charts. The disadvantage is that two charts might need more space than one chart.



### Solution 2: Indexed charts

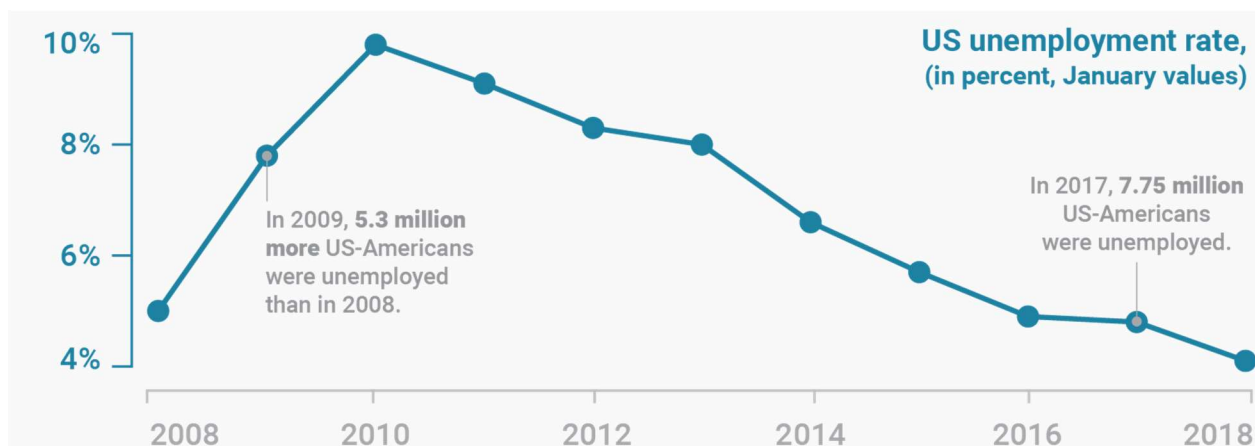
If we want to keep both data series in one chart, we can create an indexed chart. That's a chart that doesn't tell us anything about absolute numbers, but **shows the relative change of our data series over time**: By what *percentage* a variable increased or decreased over time. Labeling or tooltips can bring back information about the absolute numbers. And one can even show more than two data series in the same chart, as happened in [this chart](#) by my co-worker Gregor, who compares the growth and decline of several cryptocurrencies with each other.

This approach works only for data series with a similar rate of change, though. Cole Nussbaumer Knafllic makes that point really well in [one of her articles](#): If one of data series changes by +10000% and the other one by just +5%, the latter line will almost be invisible.



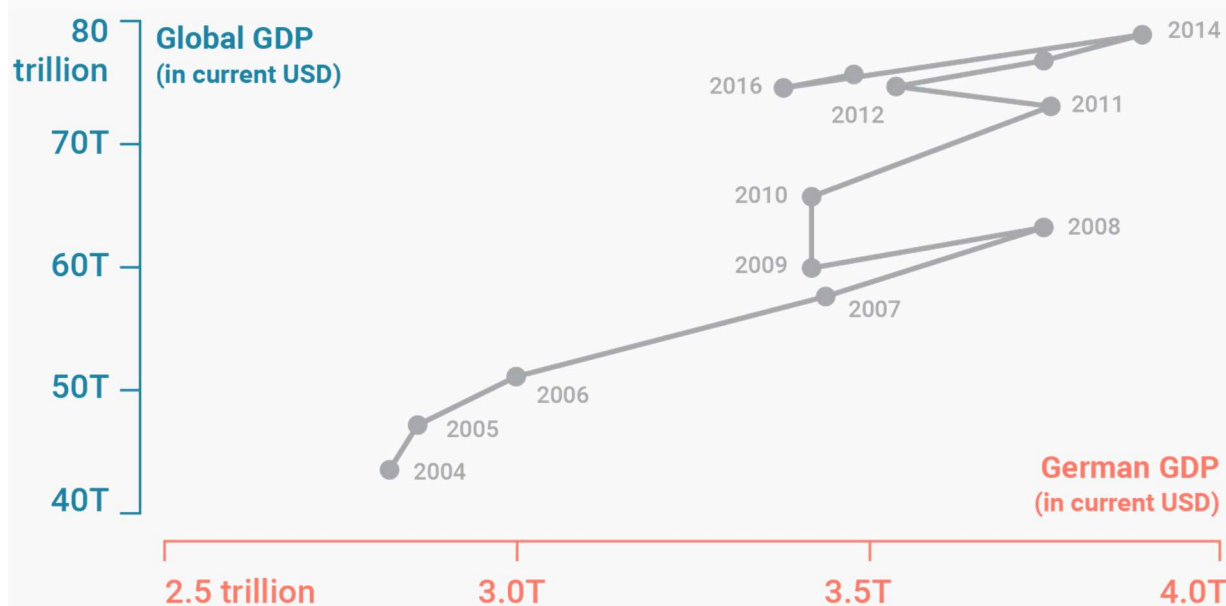
### Solution 3: Prioritizing & labeling

The third idea to prevent a dual axis is to just show one line: the more important data series of the two. We can then use chart annotations to add information about the data we leave out (the other data series). That's also a [recommendation by Cole Nussbaumer Knafllic](#), although she called it “not exactly the eloquent solution I was imagining”. Indeed, this solution won't work well for most data (including ours), but **can be a great alternative for dual axis charts that present absolute and relative numbers of the same measure**. For example, the following chart shows the unemployment *rate* in the US, but gives information about the absolute numbers in form of annotations:





### Solution 4: Connected Scatterplot

Here it gets fancy: A connected scatterplot keeps one variable on the y-axis – but instead of time, it places the second variable on the x-axis. Suddenly, time doesn't move from left to right, but wiggles through space. It's really unintuitive<sup>[2]</sup>, but also lots of fun. [Stephanie Evergreen recommends it as an alternative to a dual axis chart](#) in one of her articles. In our case, a connected scatterplot might be overkill; but we've seen [cases](#) in which they are the best chart type for showing an insight.



Of the four alternatives we show here, the first two will be useful in most cases. Let us know if we missed anything! We hope we could make our concerns understandable and show some ways how you can visualise your data even without a dual axis chart.

1. Here are articles by smart people who have opinions about using dual axis charts:  
[Dual-Scaled Axes in Graphs Are They Ever the Best Solution?](#) by Stephen Few. Looks first at column charts, then at line charts with dual axes and concludes that he "cannot think of a situation that warrants them in light of other, better solutions."  
[Two Alternatives to Using a Second Y-Axis](#) by Stephanie Evergreen. Explains two alternatives to a second Y-axis, two side-by-side graphs and a connected scatterplot.  
[Be gone, dual y-axis!](#) by Cole Nussbaumer Knaflic. Shows a case when indexed charts as an alternative for dual axis charts fail, and suggests a labeled chart instead.  
[Hadley Wickham's arguments against dual axis charts](#) on StackOverflow. Gives four arguments why it's not possible to create dual axis charts with his charting library ggplot2.  
[Dual axes time series plots may be ok sometimes after all](#) by Peter Ellis. Makes arguments against side-by-side charts, indexed charts, and connected scatterplots, and explains Do's and Don'ts of creating dual axis charts. 
2. Note that both the dual axis chart and the connected scatterplots are not intuitive, but differently so: The dual axis chart **promises the reader to be easily decipherable**, since it looks like the common line-chart that readers have seen so often and learned how to read. Quickly glancing at a dual axis chart for a second can plant misleading statements in a reader's mind. Glancing at a connected scatterplot, on the other side, just ends in a confused face and the realisation "I need to take my time to understand this chart". It's not intuitive, but it also doesn't lead to intuitive (and wrong) insights. 

# Visualization Formats, Types, and Table Structures

## Visualization Formats and Types Overview

AppCenter supports a variety of built-in visualizations formats and types based on specific table structures for apps that run once or on schedule and SQL scripts. In addition, you can upload your own visualization assets using the **Custom (Upload)** option when creating the app or uploading the script.

## Built-In Visualization Formats and Types

AppCenter supports the following built-in visualization formats and types for apps that run once or on schedule and SQL scripts:

Visualization Format	Visualization Types Supported	Required Table Structure and Example															
CFilter	<ul style="list-style-type: none"><li>• Chord</li><li>• Sigma</li><li>• Bar</li><li>• Pie</li><li>• Line</li><li>• Statistics Line</li></ul>	<div>coll_item1 - varchar coll_item2 - varchar cntb - int cnt1 - int cnt2 - int</div> <div>Example:</div> <table><thead><tr><th>coll_item1</th><th>coll_item2</th><th>cntb</th><th>cnt1</th><th>cnt2</th></tr></thead><tbody><tr><td>juice</td><td>soda</td><td>974</td><td>10428</td><td>10341</td></tr><tr><td>crackers</td><td>pretzels</td><td>957</td><td>10467</td><td>10554</td></tr></tbody></table>	coll_item1	coll_item2	cntb	cnt1	cnt2	juice	soda	974	10428	10341	crackers	pretzels	957	10467	10554
coll_item1	coll_item2	cntb	cnt1	cnt2													
juice	soda	974	10428	10341													
crackers	pretzels	957	10467	10554													
nPath®	<ul style="list-style-type: none"><li>• Hierarchical clustering tree</li><li>• Sigma</li><li>• Tree</li><li>• Sankey</li></ul>	<div>cnt - int path - varchar</div> <div>Example:</div> <table><thead><tr><th>cnt</th><th>path</th></tr></thead><tbody><tr><td>25</td><td>[empty cart,eggs,butter,salsa,chips,soda,sugar]</td></tr><tr><td>5</td><td>[empty cart,eggs,diapers,sugar,egg whites,soda,flour]</td></tr></tbody></table>	cnt	path	25	[empty cart,eggs,butter,salsa,chips,soda,sugar]	5	[empty cart,eggs,diapers,sugar,egg whites,soda,flour]									
cnt	path																
25	[empty cart,eggs,butter,salsa,chips,soda,sugar]																
5	[empty cart,eggs,diapers,sugar,egg whites,soda,flour]																
Sessionize	<ul style="list-style-type: none"><li>• Bar</li><li>• Pie</li><li>• Line</li><li>• Statistics Line</li></ul>	<div>x - int y - int</div> <div>Example:</div> <table><thead><tr><th>x</th><th>y</th></tr></thead><tbody><tr><td>1110000</td><td>12</td></tr><tr><td>1120000</td><td>23</td></tr></tbody></table>	x	y	1110000	12	1120000	23									
x	y																
1110000	12																
1120000	23																
Tfidf	<ul style="list-style-type: none"><li>• Wordcloud</li><li>• Wordbubbles wordcloud</li></ul>	<div>term - varchar tf_idf - real</div> <div>Example:</div>															

Visualization Format	Visualization Types Supported	Required Table Structure and Example									
		<table><tr><th>term</th><th>tf_idf</th></tr><tr><td>Test</td><td>1.080000809</td></tr><tr><td>Hello</td><td>12.87534987934</td></tr></table>	term	tf_idf	Test	1.080000809	Hello	12.87534987934			
term	tf_idf										
Test	1.080000809										
Hello	12.87534987934										
Find Named Entity	<ul style="list-style-type: none"><li>• Wordcloud</li><li>• Wordbubbles wordcloud</li></ul>	<div><div>id - int ENTITY - varchar TYPE - varchar</div><div>Example:<table><tr><th>id</th><th>ENTITY</th><th>TYPE</th></tr><tr><td>1</td><td>John Smith</td><td>person</td></tr><tr><td>2</td><td>Tennessee</td><td>location</td></tr></table></div></div>	id	ENTITY	TYPE	1	John Smith	person	2	Tennessee	location
id	ENTITY	TYPE									
1	John Smith	person									
2	Tennessee	location									
Forest Drive	<ul style="list-style-type: none"><li>• Dtree</li></ul>	<div><div>worker_ip - varchar task_index - integer tree_num - integer tree - varchar</div><div>Example:<table><tr><th>worker_ip</th><th>task_index</th><th>tree_num</th><th>tree</th></tr><tr><td>127.0.0.1</td><td>11</td><td>0</td><td>{"label_": "4""size_": 2"id_": 1"maxDepth_": 0"nodeType_": "CLASSIFICATION_LEAF"}</td></tr></table></div></div>	worker_ip	task_index	tree_num	tree	127.0.0.1	11	0	{"label_": "4""size_": 2"id_": 1"maxDepth_": 0"nodeType_": "CLASSIFICATION_LEAF"}	
worker_ip	task_index	tree_num	tree								
127.0.0.1	11	0	{"label_": "4""size_": 2"id_": 1"maxDepth_": 0"nodeType_": "CLASSIFICATION_LEAF"}								

If you select a **Bar** visualization type with the **Default** format, AppCenter provides the following options to customize the resulting chart:

- X-Axis (horizontal) and Y-Axis (vertical) labels
- Show Legend (identify data in visualization)
- Color Scheme (vivid, natural, cool, fire, solar, air, and more)

## Visualization Code Example for SQL Scripts

The code for each visualization type must include `--name=type`, where *type* can be anything, followed by the SQL. For example:

```
--name=bar1
select original_service, complete from appcenter_user.sdabc;

--name=line1
select original_service, complete from appcenter_user.sdabc;

--name=wordcloud1
select original_service as term, complete as tf_idf from appcenter_user.sdabc;

--name=bar2
```

# 3.0 Statistical Techniques

# Scatterplots and Correlation

**Source:** The Basic Practice of Statistics (6th ed.). Diana Mindrila, Ph.D.  
Phoebe Balentyne, M.Ed.

## Concepts:

- Displaying Relationships: Scatterplots
- Interpreting Scatterplots
- Adding Categorical Variables to Scatterplots
- Measuring Linear Association: Correlation
- Facts About Correlation

## Objectives:

- Construct and interpret scatterplots.
- Add categorical variables to scatterplots.
- Calculate and interpret correlation.
- Describe facts about correlation.

## References:

Moore, D. S., Notz, W. I., & Flinger, M. A. (2013). *The basic practice of statistics* (6<sup>th</sup> ed.). New York, NY: W. H. Freeman and Company.

## Scatterplot

- The most useful graph for displaying the relationship between two quantitative variables is a **scatterplot**.

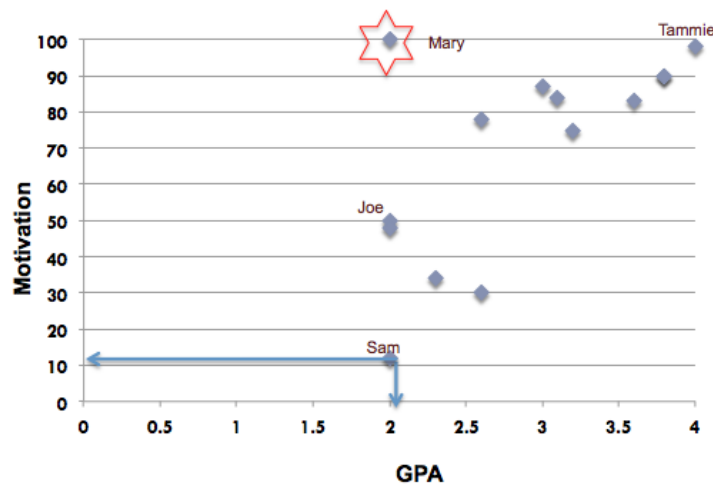
A **scatterplot** shows the relationship between two quantitative variables measured for the same individuals. The values of one variable appear on the horizontal axis, and the values of the other variable appear on the vertical axis. Each individual in the data appears as a point on the graph.

- Many research projects are **correlational studies** because they investigate the relationships that may exist between variables. Prior to investigating the relationship between two quantitative variables, it is always helpful to create a graphical representation that includes both of these variables. Such a graphical representation is called a **scatterplot**.

## Scatterplot Example

What is the relationship between students' achievement motivation and GPA?

Student	Student GPA	Motivation
Joe	2.0	50
Lisa	2.0	48
Mary	2.0	100
Sam	2.0	12
Deana	2.3	34
Sarah	2.6	30
Jennifer	2.6	78
Gregory	3.0	87
Thomas	3.1	84
Cindy	3.2	75
Martha	3.6	83
Steve	3.8	90
Jamell	3.8	90
Tammie	4.0	98



- In this example, the relationship between students' achievement motivation and their GPA is being investigated.
- The table on the left includes a small group of individuals for whom GPA and scores on a motivation scale have been recorded. GPAs can range from 0 to 4 and motivation scores in this example range from 0 to 100. Individuals in this table were ordered based on their GPA.
- Simply looking at the table shows that, in general, as GPA increases, motivation scores also increase.
- However, with a real set of data, which may have hundreds or even thousands of individuals, a pattern cannot be detected by simply looking at the numbers. Therefore, a very useful strategy is to represent the two variables graphically to illustrate the relationship between them.
- A graphical representation of individual scores on two variables is called a **scatterplot**.
- The image on the right is an example of a scatterplot and displays the data from the table on the left. GPA scores are displayed on the horizontal axis and motivation scores are displayed on the vertical axis.
- Each dot on the scatterplot represents one individual from the data set. The location of each point on the graph depends on both the GPA and motivation scores. Individuals with higher GPAs are located further to the right and individuals with higher motivation scores are located higher up on the graph.
- Sam, for example, has a GPA of 2 so his point is located at 2 on the right. He also has a motivation score of 12, so his point is located at 12 going up.
- Scatterplots are not meant to be used in great detail because there are usually hundreds of individuals in a data set.

- The purpose of a scatterplot is to provide a general illustration of the relationship between the two variables.
- In this example, in general, as GPA increases so does an individual's motivation score.
- One of the students in this example does not seem to follow the general pattern: Mary. She is one of the students with the lowest GPA, but she has the maximum score on the motivation scale. This makes her an exception or an outlier.

## Interpreting Scatterplots

### How to Examine a Scatterplot

As in any graph of data, look for the *overall pattern* and for striking *departures* from that pattern.

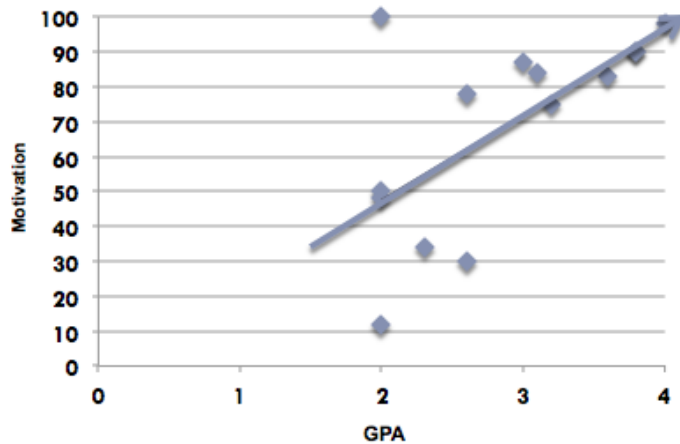
- The overall pattern of a scatterplot can be described by the **direction**, **form**, and **strength** of the relationship.
- An important kind of departure is an **outlier**, an individual value that falls outside the overall pattern of the relationship.

## Interpreting Scatterplots: Direction

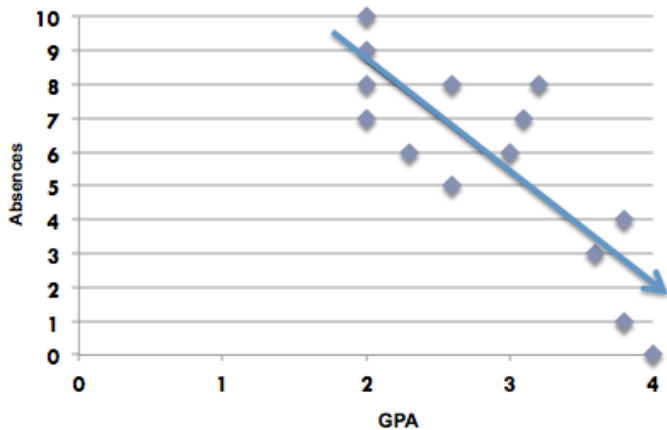
- One important component to a scatterplot is the **direction** of the relationship between the two variables.

Two variables have a **positive association** when above-average values of one tend to accompany above-average values of the other, and when below-average values also tend to occur together.

Two variables have a **negative association** when above-average values of one tend to accompany below-average values of the other.



This example compares students' achievement motivation and their GPA. These two variables have a **positive association** because as GPA increases, so does motivation.

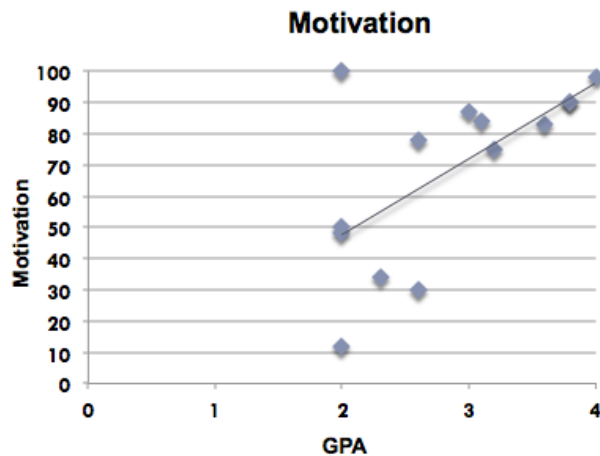


This example compares students' GPA and their number of absences. These two variables have a **negative association** because, in general, as a student's number of absences decreases, their GPA increases.

## Interpreting Scatterplots: Form

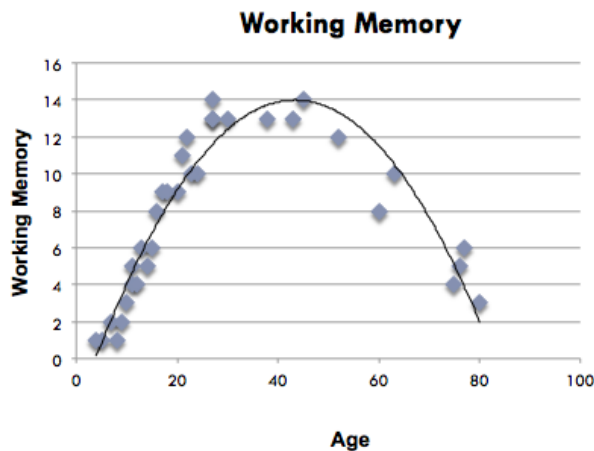
- Another important component to a scatterplot is the **form** of the relationship between the two variables.

### Linear relationship:



This example illustrates a linear relationship. This means that the points on the scatterplot closely resemble a straight line. A relationship is linear if one variable increases by approximately the same rate as the other variables changes by one unit.

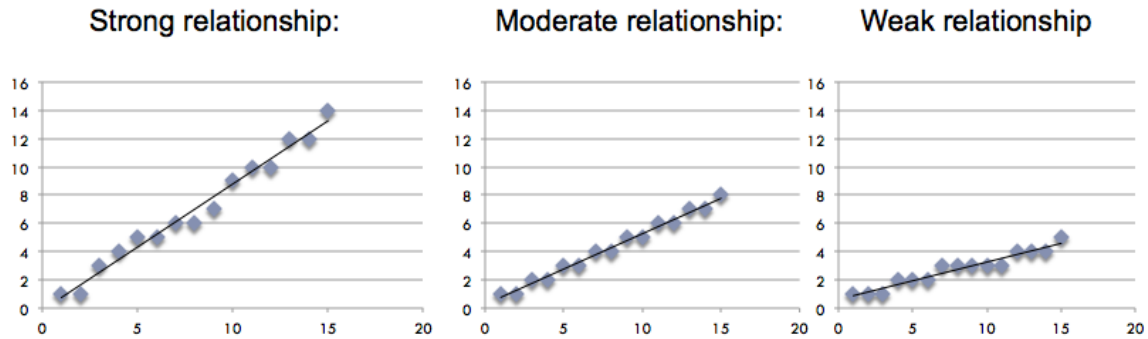
### Curvilinear relationship:



This example illustrates a relationship that has the form of a curve, rather than a straight line. This is due to the fact that one variable does not increase at a constant rate and may even start decreasing after a certain point. This example describes a curvilinear relationship between the variable "age" and the variable "working memory." In this example, working memory increases throughout childhood, remains steady in adulthood, and begins decreasing around age 50.

## Interpreting Scatterplots: Strength

- Another important component to a scatterplot is the **strength** of the relationship between the two variables.
- The **slope** provides information on the strength of the relationship.



- The strongest linear relationship occurs when the slope is 1. This means that when one variable increases by one, the other variable also increases by the same amount. This line is at a 45 degree angle.
- The strength of the relationship between two variables is a crucial piece of information. Relying on the interpretation of a scatterplot is too subjective. More precise evidence is needed, and this evidence is obtained by computing a coefficient that measures the strength of the relationship under investigation.

## Measuring Linear Association

- A scatterplot *displays* the strength, direction, and form of the relationship between two quantitative variables.
- A correlation coefficient *measures* the strength of that relationship.

The **correlation  $r$**  measures the strength of the linear relationship between two quantitative variables.

Pearson  $r$ :

$$r = \frac{1}{n-1} \sum \left( \frac{x_i - \bar{x}}{s_x} \right) \left( \frac{y_i - \bar{y}}{s_y} \right)$$

- $r$  is always a number between -1 and 1.
  - $r > 0$  indicates a positive association.
  - $r < 0$  indicates a negative association.
  - Values of  $r$  near 0 indicate a very weak linear relationship.
  - The strength of the linear relationship increases as  $r$  moves away from 0 toward -1 or 1.
  - The extreme values  $r = -1$  and  $r = 1$  occur only in the case of a perfect linear relationship.
- Calculating a Pearson correlation coefficient requires the assumption that the relationship between the two variables is linear.
  - There is a rule of thumb for interpreting the strength of a relationship based on its  $r$  value (use the absolute value of the  $r$  value to make all values positive):

### Absolute Value of $r$

$r < 0.3$

$0.3 < r < 0.5$

$0.5 < r < 0.7$

$r > 0.7$

### Strength of Relationship

None or very weak

Weak

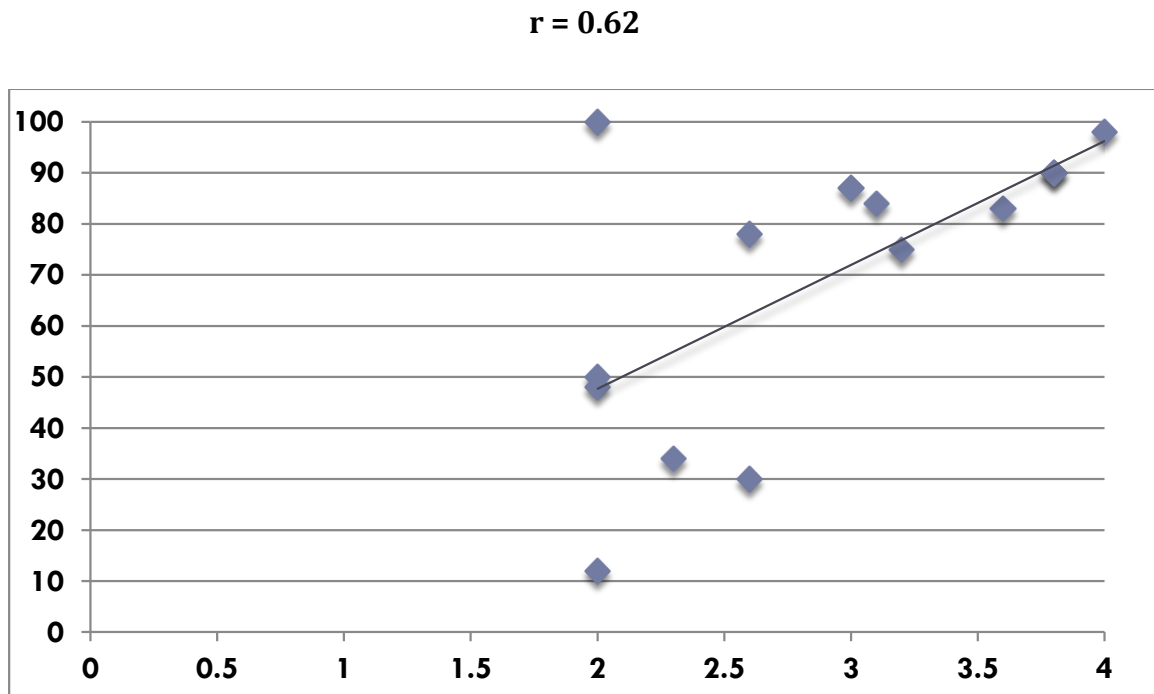
Moderate

Strong

- The relationship between two variables is generally considered strong when their  $r$  value is larger than 0.7.

## Correlations

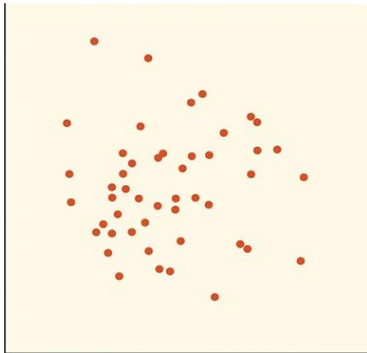
Example: There is a moderate, positive, linear relationship between GPA and achievement motivation.



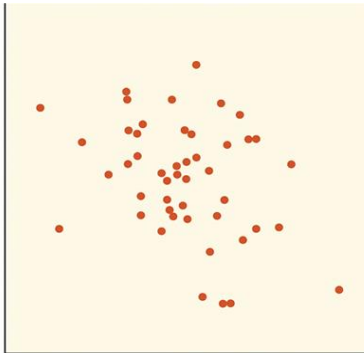
- Based on the criteria listed on the previous page, the value of  $r$  in this case ( $r = 0.62$ ) indicates that there is a positive, linear relationship of **moderate** strength between achievement motivation and GPA.

## Correlation

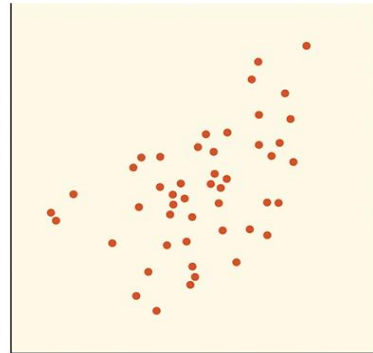
- The images below illustrate what the relationships might look like at different degrees of strength (for different values of  $r$ ).



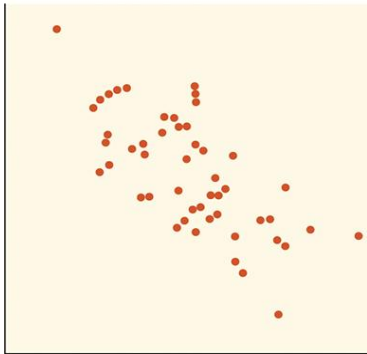
Correlation  $r = 0$



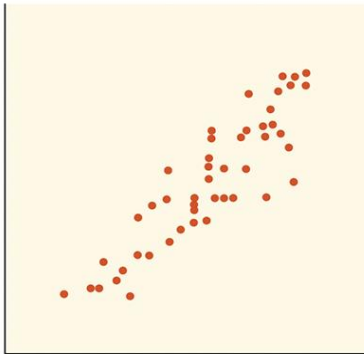
Correlation  $r = -0.3$



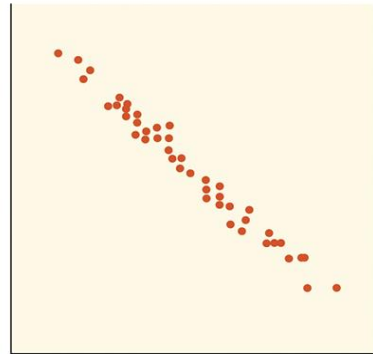
Correlation  $r = 0.5$



Correlation  $r = -0.7$



Correlation  $r = 0.9$



Correlation  $r = -0.99$

- For a correlation coefficient of zero, the points have no direction, the shape is almost round, and a line does not fit to the points on the graph.
- As the correlation coefficient increases, the observations group closer together in a linear shape.
- The line is difficult to detect when the relationship is weak (e.g.,  $r = -0.3$ ), but becomes more clear as relationships become stronger (e.g.,  $r = -0.99$ )

## Correlation Coefficients

### The Statistical Significance of Correlation Coefficients:

- Correlation coefficients have a probability (p-value), which shows **the probability that the relationship between the two variables is equal to zero** (null hypotheses; no relationship).
- **Strong** correlations have **low** p-values because the probability that they have no relationship is very low.
- Correlations are typically considered statistically significant if the p-value is lower than 0.05 in the social sciences, but the researcher has the liberty to decide the p-value for which he or she will consider the relationship to be significant.
- The value of p for which a correlation will be considered statistically significant is called the **alpha level** and must be reported.
- SPSS notation for p values: Sig. (2 tailed)

In the previous example,  $r = 0.62$  and  $p\text{-value} = 0.03$ . The p-value of 0.03 is less than the acceptable alpha level of 0.05, meaning the correlation is statistically significant.

Four things must be reported to describe a relationship:

- 1) The **strength** of the relationship given by the correlation coefficient.
- 2) The **direction** of the relationship, which can be positive or negative based on the sign of the correlation coefficient.
- 3) The **shape** of the relationship, which must always be linear to compute a Pearson correlation coefficient.
- 4) Whether or not the relationship is **statistically significant**, which is based on the p-value.

## **Facts About Correlation**

- 1) The order of variables in a correlation is not important.
- 2) Correlations provide evidence of association, not causation.
- 3)  $r$  has no units and does not change when the units of measure of  $x$ ,  $y$ , or both are changed.
- 4) Positive  $r$  values indicate positive association between the variables, and negative  $r$  values indicate negative associations.
- 5) The correlation  $r$  is always a number between -1 and 1.

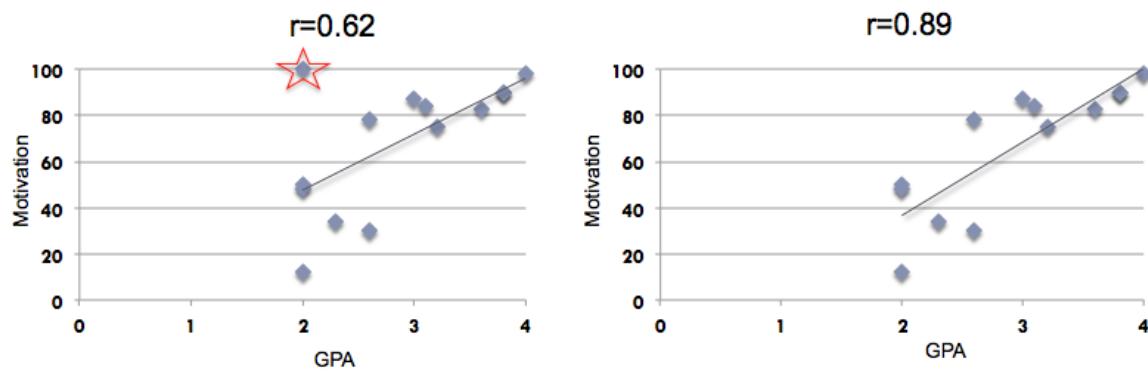
## Pearson $r$ : Assumptions

Assumptions:

- Correlation requires that both variables be quantitative.
- Correlation describes *linear* relationships. Correlation does not describe curve relationships between variables, no matter how strong the relationship is.

Cautions:

- Correlation is not resistant.  $r$  is strongly affected by outliers.
- Correlation is not a complete summary of two-variable data.
- For example:



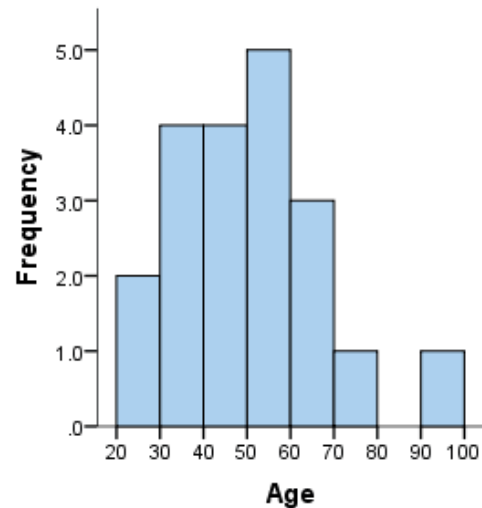
- The correlation coefficient is based on means and standard deviations, so it is not robust to outliers; it is strongly affected by extreme observations. These individuals are sometimes referred to as *influential observations* because they have a strong impact on the correlation coefficient.
- For instance, in the above example the correlation coefficient is 0.62 on the left when the outlier is included in the analysis. However, when this outlier is removed, the correlation coefficient increases significantly to 0.89.
- This one case, when included in the analysis, reduces a strong relationship to a moderate relationship.
- This case makes such a big difference in this example because the data set contains a very small number of individuals. As a general rule, as the size of the sample increases, the influence of extreme observations decreases.
- When describing the relationship between two variables, correlations are just one piece of the puzzle. This information is necessary, but not sufficient. Other analyses should also be conducted to provide more information.

# Histograms

**Source:** <https://statistics.laerd.com/statistical-guides/understanding-histograms.php>

## What is a histogram?

A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. An example of a histogram, and the raw data it was constructed from, is shown below:



36	25	38	46	55	68	72	55	36	38
67	45	22	48	91	46	52	61	58	55

## How do you construct a histogram from a continuous variable?

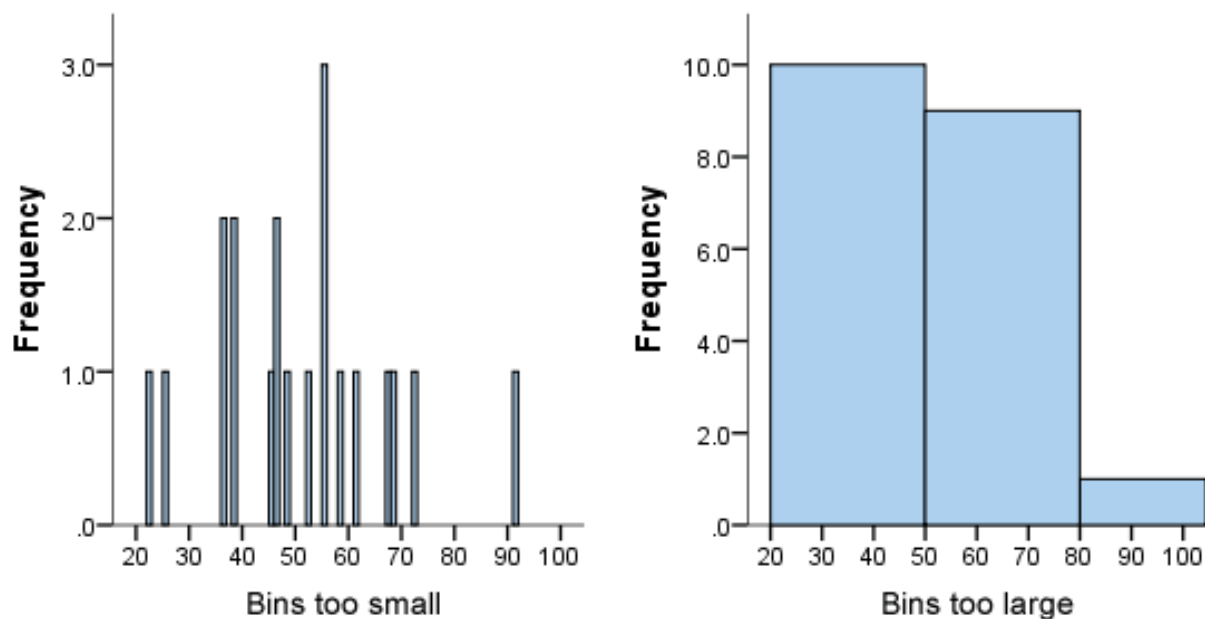
To construct a histogram from a continuous variable you first need to split the data into intervals, called **bins**. In the example above, **age** has been split into bins, with each bin representing a 10-year period starting at 20 years. Each bin contains the number of occurrences of scores in the data set that are contained within that bin. For the above data set, the frequencies in each bin have been tabulated along with the scores that contributed to the frequency in each bin (see below):

Bin	Frequency	Scores Included in Bin
20-30	2	25,22
30-40	4	36,38,36,38
40-50	4	46,45,48,46
50-60	5	55,55,52,58,55
60-70	3	68,67,61
70-80	1	72
80-90	0	-
90-100	1	91

Notice that, unlike a bar chart, there are no "gaps" between the bars (although some bars might be "absent" reflecting no frequencies). This is because a histogram represents a continuous data set, and as such, there are no gaps in the data (although you will have to decide whether you round up or round down scores on the boundaries of bins).

### Choosing the correct bin width

There is no right or wrong answer as to how wide a bin should be, but there are rules of thumb. You need to make sure that the bins are not too small or too large. Consider the histogram we produced earlier (see above): the following histograms use the same data, but have either much smaller or larger bins, as shown below:



We can see from the histogram on the left that the bin width is too small because it shows too much individual data and does not allow the underlying pattern (frequency distribution) of the data to be easily seen. At the other end of the scale is the diagram on the right, where the bins are too large, and again, we are unable to find the underlying trend in the data.

### Histograms are based on area, not height of bars

In a histogram, it is the area of the bar that indicates the frequency of occurrences for each bin. This means that the height of the bar does not necessarily indicate how many occurrences of scores there were within each individual bin. It is the product of height multiplied by the width of the bin that indicates the frequency of occurrences within that bin. One of the reasons that the height of the bars is

---

often incorrectly assessed as indicating frequency and not the area of the bar is due to the fact that a lot of histograms often have equally spaced bars (bins), and under these circumstances, the height of the bin does reflect the frequency.

**What is the difference between a bar chart and a histogram?**

The major difference is that a histogram is only used to plot the frequency of score occurrences in a continuous data set that has been divided into classes, called bins. Bar charts, on the other hand, can be used for a great deal of other types of variables including ordinal and nominal data sets.

# Assumption of Linearity

**Source:** <http://people.duke.edu/~rnau/testing.htm>

There are **four principal assumptions** which justify the use of linear regression models for purposes of inference or prediction:

**(i) linearity and additivity** of the relationship between dependent and independent variables:

(a) The expected value of dependent variable is a straight-line function of each independent variable, holding the others fixed.

(b) The slope of that line does not depend on the values of the other variables.

(c) The effects of different independent variables on the expected value of the dependent variable are additive.

**(ii) statistical independence** of the errors (in particular, no correlation between consecutive errors in the case of time series data)

**(iii) homoscedasticity** (constant variance) of the errors

(a) versus time (in the case of time series data)

(b) versus the predictions

(c) versus any independent variable

**(iv) normality** of the error distribution.

If any of these assumptions is violated (i.e., if there are nonlinear relationships between dependent and independent variables or the errors exhibit correlation, heteroscedasticity, or non-normality), then the forecasts, confidence intervals, and scientific insights yielded by a regression model may be (at best) inefficient or (at worst) seriously biased or misleading.

**Violations of linearity or additivity** are extremely serious: if you fit a linear model to data which are nonlinearly or nonadditively related, your predictions are likely to be seriously in error, especially when you extrapolate beyond the range of the sample data.

**How to diagnose:** nonlinearity is usually most evident in a plot of **observed versus predicted values** or a plot of **residuals versus predicted values**, which are a part of standard regression output. The points should be symmetrically distributed around a diagonal line in the former plot or around horizontal line in the latter plot, with a roughly constant variance. (The residual-versus-predicted-plot is better than the observed-versus-predicted plot for this purpose, because it eliminates the visual distraction of a sloping pattern.) Look carefully for evidence of a "bowed" pattern, indicating that the model makes systematic errors whenever it is making unusually large or small predictions. In multiple regression models,

nonlinearity or nonadditivity may also be revealed by systematic patterns in plots of the **residuals versus individual independent variables**.

**How to fix:** consider applying a *nonlinear transformation* to the dependent and/or independent variables *if* you can think of a transformation that seems appropriate. For example, if the data are strictly positive, the log transformation is an option. (The logarithm base does not matter--all log functions are same up to linear scaling--although the natural log is usually preferred because small changes in the natural log are equivalent to percentage changes. If a log transformation is applied to the dependent variable only, this is equivalent to assuming that it grows (or decays) exponentially as a function of the independent variables. If a log transformation is applied to *both* the dependent variable and the independent variables, this is equivalent to assuming that the effects of the independent variables are *multiplicative* rather than additive in their original units. This means that, on the margin, a small *percentage* change in one of the independent variables induces a proportional *percentage* change in the expected value of the dependent variable, other things being equal.

Another possibility to consider is adding *another regressor* that is a nonlinear function of one of the other variables. For example, if you have regressed Y on X, and the graph of residuals versus predicted values suggests a parabolic curve, then it may make sense to regress Y on both X and  $X^2$  (i.e., X-squared). The latter transformation is possible even when X and/or Y have negative values, whereas logging is not. Higher-order terms of this kind (cubic, etc.) might also be considered in some cases. But don't get carried away! This sort of "polynomial curve fitting" can be a nice way to draw a smooth curve through a wavy pattern of points (in fact, it is a trend-line option on scatterplots on Excel), but it is usually a terrible way to extrapolate outside the range of the sample data.

Finally, it may be that you have overlooked some *entirely different independent variable* that explains or corrects for the nonlinear pattern or interactions among variables that you are seeing in your residual plots. In that case the shape of the pattern, together with economic or physical reasoning, may suggest some likely suspects. For example, if the strength of the linear relationship between Y and  $X_1$  depends on the level of some other variable  $X_2$ , this could perhaps be addressed by creating a new independent variable that is the product of  $X_1$  and  $X_2$ . In the case of time series data, if the trend in Y is believed to have changed at a particular point in time, then the addition of a *piecewise linear* trend variable (one whose string of values looks like 0, 0, ..., 0, 1, 2, 3, ...) could be used to fit the kink in the data. Such a variable can be considered as the product of a trend variable and a dummy variable. Again, though, you need to beware of overfitting the sample data by throwing in artificially constructed variables that are poorly motivated. At the end of the day you need to be able to interpret the model and explain (or sell) it to others.

# UnivariateStatistics (ML Engine)

The UnivariateStatistics function calculates descriptive statistics for a set of target columns.

## UnivariateStatistics Syntax

### Version 1.2

```
SELECT * FROM UnivariateStatistics (  
  ON { table | view | (query) } AS InputTable  
  [ OUT TABLE MomentsTableName (moments_table_name) ]  
  [ OUT TABLE BasicTableName (basic_table_name) ]  
  [ OUT TABLE QuantilesTableName (quantiles_table_name) ]  
  USING  
  [ TargetColumns ('target_column' [...]) |  
    ExcludeColumns ('exclude_column' [...])  
  ]  
  [ PartitionColumns ('partition_column' [...]) ]  
  [ StatisticsGroups ('statistics_group' [...]) ]  
) AS alias;
```

## UnivariateStatistics Syntax Elements

### MomentsTableName

[Required if you omit StatisticsGroups syntax element or specify 'moments'.] Specify the name for the output table that contains the moments.

### BasicTableName

[Required if you omit StatisticsGroups or specify 'basic'.] Specify the name for the output table that contains the basic statistics.

### QuantilesTableName

[Required if you omit StatisticsGroups or specify 'quantiles'.] Specify the name for the output table that contains the quantiles.

### TargetColumns

[Optional] Specify the names of the InputTable columns for which to compute statistics.

Default: All numerical InputTable columns

### ExcludeColumns

[Optional] Specify the names of the InputTable columns to exclude from statistics calculation.

**PartitionColumns**

[Optional] Specify the names of the InputTable columns on which to partition the input. The function copies these columns to the output table.

Default behavior: The function treats all rows as a single partition.

**StatisticsGroups**

[Optional] Specify the group or groups of statistics to calculate:

<i>statistics_group</i>	Statistics to Calculate
'moments '	<ul style="list-style-type: none"> <li>• Number of observations</li> <li>• Sum</li> <li>• Mean</li> <li>• Variance</li> <li>• Standard deviation</li> <li>• Standard error</li> <li>• Skewness</li> <li>• Kurtosis</li> <li>• Coefficient of variation</li> <li>• Uncorrected sum of squares</li> <li>• Corrected sum of squares</li> </ul>
'basic '	<ul style="list-style-type: none"> <li>• Number of observations</li> <li>• Number of NULL values</li> <li>• Number of positive, negative, and zero values</li> <li>• Number of unique values</li> <li>• Mode</li> <li>• Median</li> <li>• Mean</li> <li>• Standard deviation</li> <li>• Variance</li> <li>• Range</li> <li>• Interquartile range</li> <li>• Harmonic mean</li> <li>• Geometric mean</li> <li>• Highest and lowest five values</li> </ul>
'quantiles '	<ul style="list-style-type: none"> <li>• Minimum and maximum values</li> <li>• 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th, and 99th percentiles</li> </ul>

Default behavior: The function calculates all three groups of statistics.

## UnivariateStatistics Input

### InputTable Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Defines a partition for statistics calculation.
<i>target_column</i>	SMALLINT, INT, BIGINT, NUMERIC, or DOUBLE PRECISION	[Column appears once for each <i>target_column</i> , which is either specified by TargetColumns or omitted from ExcludeColumns. ] Column for which to compute statistics. At least one <i>target_column</i> must be numeric.

## UnivariateStatistics Output

### Output Message Schema

Column	Data Type	Description
message	VARCHAR	Reports whether function succeeded and saved output files.

### MomentsStatistics, BasicStatistics, and QuantileStatistics Schema

The StatisticsGroups syntax element determines which statistics tables the function outputs:

- If you omit StatisticsGroups, the function calculates all statistics and outputs three tables, one for each *statistics\_group*.
- If you specify StatisticsGroups, the function outputs a table for each specified *statistics\_group*.

For each table to be output, you must specify a name, using the MomentsTableName, BasicTableName, or QuantilesTableName syntax element.

The tables have the same schema.

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	[Column appears once for each specified <i>partition_column</i> .] Column copied from input table. Defines a partition for statistics calculation.
stats	VARCHAR	Identifies statistic in row; for example, "Coefficient of variation" or "Corrected sum of squares."
<i>result_value</i>	DOUBLE PRECISION	[Column appears once for each specified <i>target_column</i> .] Calculated statistic identified by statistics column.

## UnivariateStatistics Examples

### UnivariateStatistics Example: ExcludeColumns, All Statistics

This example excludes columns id and period from the target columns and outputs all three statistics tables, by default.

#### Input

- finance\_data3, as in [VARMAX Example: No Exogenous Model](#)

#### SQL Call

```
DROP TABLE moments;
DROP TABLE basic;
DROP TABLE quantiles;

SELECT * FROM UnivariateStatistics (
ON finance_data3 AS InputTable
OUT TABLE MomentsTableName(moments)
OUT TABLE BasicTableName(basic)
OUT TABLE QuantilesTableName(quantiles)
USING
ExcludeColumns('id','period')
) AS dt ;
```

#### Output

message

-----  
UnivariateStatistics succeeded. The output tables are saved.

```
SELECT * FROM moments;
```

stats	expenditure	income	investment
-----	-----	-----	-----
Standard deviation	590.923585337053	698.928750981727	210.746691977944
Corrected sum of squares	3.17763522173913E7	4.44536273043478E7	
4041689.30434783			
Skewness	0.473364666302052	0.446362736743103	0.422507147168344
Number of observations	92.0	92.0	92.0
Kurtosis	-1.15712585537242	-1.1624376180194	-1.001282169732
Variance	349190.683707597	488501.398948877	44414.1681796464
Uncorrected sum of squares	1.5700013E8	2.13389608E8	2.4530266E7
Coefficient of variation	0.506502784308876	0.515781476323667	
0.446579502072297			

Standard error	61.6080425745307	72.868359489299	21.9718614961239
Mean	1166.67391304348	1355.08695652174	471.913043478261
Sum	107334.0	124668.0	43416.0

SELECT \* FROM basic ORDER BY 1;

stats	expenditure	income	investment
-----	-----	-----	-----
Bottom 5 (1)	415.0	451.0	179.0
Bottom 5 (2)	421.0	465.0	180.0
Bottom 5 (3)	434.0	485.0	185.0
Bottom 5 (4)	448.0	493.0	192.0
Bottom 5 (5)	458.0	509.0	202.0
Geometric mean	1020.53565750432	1176.50711695451	425.089198191843
Harmonic mean	891.824091552794	1017.9035289329	381.44257427176
Interquartile range	997.0	1159.0	311.0
Mean	1166.67391304348	1355.08695652174	471.913043478261
Median	1013.0	1178.0	494.0
Mode	574.0	799.0	519.0
Number of negative values	0.0	0.0	0.0
Number of NULL values	0.0	0.0	0.0
Number of positive values	92.0	92.0	92.0
Number of unique values	91.0	91.0	83.0
Number of zero values	0.0	0.0	0.0
Range	1856.0	2200.0	691.0
Standard deviation	590.923585337053	698.928750981727	210.746691977944
Top 5 (1)	2271.0	2651.0	870.0
Top 5 (2)	2250.0	2639.0	860.0
Top 5 (3)	2237.0	2628.0	853.0
Top 5 (4)	2235.0	2620.0	852.0
Top 5 (5)	2225.0	2618.0	844.0
Variance	349190.683707597	488501.398948877	44414.1681796464

SELECT \* FROM quantiles;

stats	expenditure	income	investment
-----	-----	-----	-----
1%	415.0	451.0	179.0
10%	497.0	548.0	214.0
25%	653.0	751.0	286.0
5%	458.0	509.0	202.0
50%	1013.0	1178.0	494.0
75%	1650.0	1910.0	597.0
90%	2102.0	2457.0	830.0
95%	2206.0	2580.0	833.0
99%	2250.0	2639.0	860.0

Maximum	2271.0	2651.0	870.0
Minimum	415.0	451.0	179.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## UnivariateStatistics Example: TargetColumns, PartitionColumns

This example specifies the target columns explicitly and partitions the data.

### Input

- finance\_data3, as in [VARMAX Example: No Exogenous Model](#)

### SQL Call

```
DROP TABLE moments;
DROP TABLE basic;
DROP TABLE quantiles;

SELECT * FROM UnivariateStatistics(
  ON finance_data3 AS InputTable
  OUT TABLE MomentsTableName(moments)
  OUT TABLE BasicTableName(basic)
  OUT TABLE QuantilesTableName(quantiles)
  USING
  TargetColumns('expenditure','income','investment')
  PartitionColumns('id')
) AS dt;
```

### Output

message

-----  
UnivariateStatistics succeeded. The output tables are saved.

```
SELECT * FROM moments;
```

id stats	expenditure	income	investment
-----	-----	-----	-----
2 Coefficient of variation	0.298644349474261	0.296222528540145	
0.255444956277228			
2 Kurtosis	-0.816037129460566	-0.858783399504378	
0.169604612557607			
1 Corrected sum of squares	5781986.4	7886360.4	666306.0
1 Standard deviation	385.0403375472	449.682520486142	

130.70871903117			
3 Corrected sum of squares	724349.666666667	1108844.91666667	
141336.916666667			
3 Standard deviation	256.612489362793	317.496587908175	
113.352594174375			
2 Mean	1355.6	1589.275	556.675
2 Uncorrected sum of squares	7.9898048E7	1.09675485E8	
1.3184093E7			
1 Number of observations	40.0	40.0	40.0
1 Skewness	2.41487088335036	2.38651478870263	
2.25360944369739			
3 Number of observations	12.0	12.0	12.0
3 Skewness	-0.779317508353147	-0.784652108192004	
-0.736147966158991			
2 Sum	54224.0	63571.0	22267.0
2 Standard error	64.0111849200731	74.4367050647558	
22.4837658670451			
1 Variance	148256.061538462	202214.369230769	
17084.7692307692			
1 Kurtosis	4.8015109257614	4.71437243981462	
4.42294797371972			
3 Variance	65849.9696969697	100804.083333333	
12848.8106060606			
3 Kurtosis	-1.54284162536314	-1.57575089282068	
-1.61592354816352			
1 Coefficient of variation	0.539121167106132	0.554342357601259	
0.434248235983952			
2 Standard deviation	404.842280147308	470.779059045639	
142.199821035626			
3 Coefficient of variation	0.125472653913842	0.132987505843069	
0.149328261070645			
1 Uncorrected sum of squares	2.6185252E7	3.4208178E7	
4290346.0			
2 Corrected sum of squares	6391993.6	8643683.975	788610.775
3 Uncorrected sum of squares	5.091683E7	6.9505945E7	
7055827.0			
1 Mean	714.2	811.2	301.0
2 Skewness	0.539127192993609	0.475410247931558	
0.806624515469788			
3 Mean	2045.16666666667	2387.41666666667	759.083333333333
1 Standard error	60.8802228844601	71.1010494350768	
20.6668631090747			
2 Number of observations	40.0	40.0	40.0
3 Standard error	74.0776449055143	91.6533702477862	

32.722075379959

1 Sum	28568.0	32448.0	12040.0
3 Sum	24542.0	28649.0	9109.0
2 Variance	163897.271794872	221632.922435897	

20220.7891025641

SELECT \* FROM basic ORDER BY 1;

id stats	expenditure	income	investment
1 Median	602.0	694.0	280.0
1 Standard deviation	385.0403375472	449.682520486142	130.70871903117
1 Top 5 (5)	779.0	897.0	322.0
1 Bottom 5 (1)	415.0	451.0	179.0
1 Number of zero values	0.0	0.0	0.0
1 Bottom 5 (5)	458.0	509.0	202.0
1 Geometric mean	652.322122537438	736.802888310562	282.108617595293
1 Harmonic mean	614.458471878887	691.093427241673	268.951437475886
1 Interquartile range	194.0	241.0	75.0
1 Top 5 (1)	1842.0	2132.0	700.0
1 Top 5 (3)	1807.0	2070.0	658.0
1 Bottom 5 (3)	434.0	485.0	185.0
1 Top 5 (4)	1774.0	2040.0	635.0
1 Number of NULL values	0.0	0.0	0.0
1 Top 5 (2)	1831.0	2121.0	675.0
1 Bottom 5 (2)	421.0	465.0	180.0
1 Variance	148256.061538462	202214.369230769	17084.7692307692
1 Mean	714.2	811.2	301.0
1 Mode	574.0	799.0	280.0
1 Number of unique values	39.0	39.0	37.0
1 Range	1427.0	1681.0	521.0
1 Bottom 5 (4)	448.0	493.0	192.0
1 Number of negative values	0.0	0.0	0.0
1 Number of positive values	40.0	40.0	40.0
2 Bottom 5 (3)	837.0	979.0	364.0
2 Top 5 (3)	2061.0	2423.0	844.0
2 Interquartile range	554.0	653.0	76.0
2 Median	1267.0	1493.0	526.0
2 Standard deviation	404.842280147308	470.779059045639	142.199821035626
2 Bottom 5 (5)	881.0	1025.0	375.0
2 Harmonic mean	1246.1032651127	1460.64283484258	524.31105430778
2 Top 5 (5)	1994.0	2318.0	816.0
2 Bottom 5 (1)	798.0	922.0	315.0
2 Geometric mean	1299.05445462922	1523.24641529642	540.054672867882
2 Number of zero values	0.0	0.0	0.0

2 Top 5 (1)	2121.0	2470.0	853.0
2 Number of positive values	40.0	40.0	40.0
2 Number of negative values	0.0	0.0	0.0
2 Range	1323.0	1548.0	538.0
2 Top 5 (4)	2056.0	2369.0	830.0
2 Number of NULL values	0.0	0.0	0.0
2 Mean	1355.6	1589.275	556.675
2 Number of unique values	40.0	40.0	36.0
2 Top 5 (2)	2102.0	2457.0	852.0
2 Bottom 5 (2)	816.0	949.0	339.0
2 Mode	NULL	NULL	519.0
2 Variance	163897.271794872	221632.922435897	20220.7891025641
2 Bottom 5 (4)	858.0	988.0	371.0
3 Number of zero values	0.0	0.0	0.0
3 Harmonic mean	2012.97784954935	2344.96421517024	741.962569222229
3 Top 5 (3)	2237.0	2628.0	833.0
3 Interquartile range	513.0	644.0	220.0
3 Standard deviation	256.612489362793	317.496587908175	113.352594174375
3 Bottom 5 (5)	2145.0	2521.0	801.0
3 Geometric mean	2029.47884297898	2366.7551731771	750.758507052472
3 Bottom 5 (1)	1650.0	1910.0	597.0
3 Top 5 (5)	2225.0	2618.0	830.0
3 Bottom 5 (3)	1722.0	1976.0	611.0
3 Top 5 (1)	2271.0	2651.0	870.0
3 Median	2164.0	2545.0	824.0
3 Variance	65849.9696969697	100804.083333333	12848.8106060606
3 Number of unique values	12.0	12.0	11.0
3 Number of negative values	0.0	0.0	0.0
3 Range	621.0	741.0	273.0
3 Number of NULL values	0.0	0.0	0.0
3 Mean	2045.16666666667	2387.41666666667	759.083333333333
3 Mode	NULL	NULL	830.0
3 Bottom 5 (2)	1685.0	1943.0	603.0
3 Top 5 (2)	2250.0	2639.0	860.0
3 Number of positive values	12.0	12.0	12.0
3 Bottom 5 (4)	1752.0	2018.0	619.0
3 Top 5 (4)	2235.0	2620.0	831.0

```
SELECT * FROM quantiles ORDER BY 1,2;
```

id	stats	expenditure	income	investment
1	1%	415.0	451.0	179.0
1	10%	448.0	493.0	192.0
1	25%	510.0	558.0	229.0

1 5%	421.0	465.0	180.0
1 50%	602.0	694.0	280.0
1 75%	704.0	799.0	304.0
1 90%	779.0	897.0	322.0
1 95%	1807.0	2070.0	658.0
1 99%	1842.0	2132.0	700.0
1 Maximum	1842.0	2132.0	700.0
1 Minimum	415.0	451.0	179.0
2 1%	798.0	922.0	315.0
2 10%	858.0	988.0	371.0
2 25%	1013.0	1178.0	494.0
2 5%	816.0	949.0	339.0
2 50%	1267.0	1493.0	526.0
2 75%	1567.0	1831.0	570.0
2 90%	1994.0	2318.0	816.0
2 95%	2061.0	2423.0	844.0
2 99%	2121.0	2470.0	853.0
2 Maximum	2121.0	2470.0	853.0
2 Minimum	798.0	922.0	315.0
3 1%	1650.0	1910.0	597.0
3 10%	1650.0	1910.0	597.0
3 25%	1722.0	1976.0	611.0
3 5%	1650.0	1910.0	597.0
3 50%	2164.0	2545.0	824.0
3 75%	2235.0	2620.0	831.0
3 90%	2250.0	2639.0	860.0
3 95%	2250.0	2639.0	860.0
3 99%	2271.0	2651.0	870.0
3 Maximum	2271.0	2651.0	870.0
3 Minimum	1650.0	1910.0	597.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## UnivariateStatistics Example: StatisticsGroups ('basic')

This example calculates and outputs only basic statistics.

### Input

- finance\_data3, as in [VARMAX Example: No Exogenous Model](#)

### SQL Call

```
DROP TABLE basic_2;
```

```

SELECT * FROM UnivariateStatistics(
  ON finance_data3 AS InputTable
  OUT TABLE BasicTableName(basic_2)
  USING
  ExcludeColumns('id','period')
  StatisticsGroups('BASIC')
) AS dt;

```

## Output

message

-----  
UnivariateStatistics succeeded. The output tables are saved.

The output table, basic\_2, is the same as [UnivariateStatistics Example: ExcludeColumns, All Statistics](#) output table basic.

```
SELECT * FROM basic_2 ORDER BY 1;
```

stats	expenditure	income	investment
Bottom 5 (1)	415.0	451.0	179.0
Bottom 5 (2)	421.0	465.0	180.0
Bottom 5 (3)	434.0	485.0	185.0
Bottom 5 (4)	448.0	493.0	192.0
Bottom 5 (5)	458.0	509.0	202.0
Geometric mean	1020.53565750432	1176.50711695451	425.089198191843
Harmonic mean	891.824091552794	1017.9035289329	381.44257427176
Interquartile range	997.0	1159.0	311.0
Mean	1166.67391304348	1355.08695652174	471.913043478261
Median	1013.0	1178.0	494.0
Mode	574.0	799.0	519.0
Number of negative values	0.0	0.0	0.0
Number of NULL values	0.0	0.0	0.0
Number of positive values	92.0	92.0	92.0
Number of unique values	91.0	91.0	83.0
Number of zero values	0.0	0.0	0.0
Range	1856.0	2200.0	691.0
Standard deviation	590.923585337053	698.928750981727	210.746691977944
Top 5 (1)	2271.0	2651.0	870.0
Top 5 (2)	2250.0	2639.0	860.0
Top 5 (3)	2237.0	2628.0	853.0
Top 5 (4)	2235.0	2620.0	852.0

Top 5 (5)	2225.0	2618.0	844.0
Variance	349190.683707597	488501.398948877	44414.1681796464

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.



Published on STAT 504 (<https://onlinecourses.science.psu.edu/stat504/>)

[Home](#) > [A Review of the Principles of Statistics](#) > [Statistical Inference and Estimation](#) > Hypothesis Testing

# Hypothesis Testing

Key Topics:

- Basic approach
- Null and alternative hypothesis
- Decision making and the  $p$ -value
- Z-test & Nonparametric alternative

## Basic approach to hypothesis testing

1. **State a model** describing the relationship between the explanatory variables and the outcome variable(s) in the population and the nature of the variability. **State all of your assumptions.**
2. **Specify the null and alternative hypotheses** in terms of the parameters of the model.
3. Invent a **test statistic** that will tend to be different under the null and alternative hypotheses.
4. Using the assumptions of step 1, **find the theoretical sampling distribution** of the statistic under the null hypothesis of step 2. Ideally the form of the sampling distribution should be one of the “standard distributions”(e.g. normal,  $t$ , binomial..)
5. **Calculate a  $p$ -value**, as the area under the sampling distribution more extreme than your statistic. Depends on the form of the alternative hypothesis.
6. **Choose your acceptable type 1 error rate** (alpha) and **apply the decision rule**: reject the null hypothesis if the  $p$ -value is less than alpha, otherwise do not reject.

## One sample z-test

1. Assume data are *independently* sampled from a *normal distribution* with unknown mean  $\mu$  and known variance  $\sigma^2$ .
2. Specify:
 

$H_0: \mu = \mu_0$		$H_A: \mu \neq \mu_0$
$H_0: \mu \leq \mu_0$	vs. one of those	$H_A: \mu > \mu_0$
$H_0: \mu \geq \mu_0$		$H_A: \mu < \mu_0$
3. Use a z-statistic:
  - $\frac{\bar{X} - \mu_0}{\sigma / \sqrt{n}}$
  - general form is: (estimate - value we are testing)/(st.dev of the estimate)
  - z-statistic follows  $N(0,1)$  distribution
4. Calculate the  $p$ -value:
  - $2 \times$  the area above  $|z|$ , area above  $z$ , or area below  $z$ , or
  - compare the statistic to a critical value,  $|z| \geq z_{\alpha/2}$ ,  $z \geq z_\alpha$ , or  $z \leq -z_\alpha$
5. Choose the acceptable level of Alpha = 0.05, we conclude .... ?

## Making the Decision

It is either **likely** or **unlikely** that we would collect the evidence we did given the initial assumption. (Note: “likely” or “unlikely” is measured by calculating a probability!)

If it is **likely**, then we “**do not reject**” our initial assumption. There is not enough evidence to do otherwise.

If it is **unlikely**, then:

- either our initial assumption is correct and we experienced an unusual event or,
- our initial assumption is incorrect

In statistics, if it is unlikely, we decide to “**reject**” our initial assumption.

---

## Example: Criminal Trial Analogy

First, state 2 hypotheses, the **null hypothesis** (“ $H_0$ ”) and the **alternative hypothesis** (“ $H_A$ ”)

- $H_0$ : Defendant is not guilty.
- $H_A$ : Defendant is guilty.

Usually the  $H_0$  is a statement of “no effect”, or “no change”, or “chance only” about a population parameter.

While the  $H_A$ , depending on the situation, is that there is a difference, trend, effect, or a relationship with respect to a population parameter.

- It can one-sided and two-sided.
- In two-sided we only care there is a difference, but not the direction of it. In one-sided we care about a particular direction of the relationship. We want to know if the value is strictly larger or smaller.

Then, collect evidence, such as finger prints, blood spots, hair samples, carpet fibers, shoe prints, ransom notes, handwriting samples, etc. (In statistics, the **data** are the evidence.)

Next, you make your initial assumption.

- Defendant is innocent until proven guilty.

In statistics, **we always assume the null hypothesis is true**.

Then, make a decision based on the available evidence.

- If there is sufficient evidence (“beyond a reasonable doubt”), **reject the null hypothesis**. (Behave as if defendant is guilty.)
- If there is not enough evidence, **do not reject the null hypothesis**. (Behave as if defendant is not guilty.)

If the observed outcome, e.g., a sample statistic, is surprising under the assumption that the null hypothesis is true, but more probable if the alternative is true, then this outcome is evidence against

$H_0$  and in favor of  $H_A$ .

An observed effect so large that it would rarely occur by chance is called statistically significant (i.e., not likely to happen by chance).

## Using the $p$ -value to make the decision

The  $p$ -value represents **how likely** we would be to observe such an extreme sample if the null hypothesis were true. The  **$p$ -value is a probability** computed assuming the null hypothesis is true, that the test statistic would take a value as extreme or more extreme than that actually observed. Since it's a probability, it is a number between 0 and 1. The closer the number is to 0 means the event is "unlikely." So if  $p$ -value is "small," (typically, less than 0.05), we can then reject the null hypothesis.

## Significance level and $p$ -value

Significance level,  $\alpha$ , is a decisive value for  $p$ -value. In this context, significant does not mean "important", but it means "not likely to happened just by chance".

$\alpha$  is the maximum probability of rejecting the null hypothesis when the null hypothesis is true. If  $\alpha = 1$  we always reject the null, if  $\alpha = 0$  we never reject the null hypothesis. In articles, journals, etc... you may read: "The results were significant ( $p < 0.05$ ).". So if  $p = 0.03$ , it's significant at the level of  $\alpha = 0.05$  but not at the level of  $\alpha = 0.01$ . If we reject the  $H_0$  at the level of  $\alpha = 0.05$  (which corresponds to 95% CI), we are saying that if  $H_0$  is true, the observed phenomenon would happen no more than 5% of the time (that is 1 in 20). If we choose to compare the  $p$ -value to  $\alpha = 0.01$ , we are insisting on a stronger evidence!

### ***Very Important Point!***

Neither decision of rejecting or not rejecting the  $H_0$  entails proving the null hypothesis or the alternative hypothesis. We merely state there is enough evidence to behave one way or the other. This is also always true in statistics!

So, what kind of error could we make? No matter what decision we make, there is always a chance we made an error.

## Errors in Criminal Trial:

	Truth	
Jury Decision	Not Guilty	Guilty
Not Guilty	OK	ERROR
Guilty	ERROR	OK

## Errors in Hypothesis Testing

**Type I error** (False positive): The null hypothesis is rejected when it is true.

- $\alpha$  is the maximum probability of making a Type I error.

**Type II error** (False negative): The null hypothesis is not rejected when it is false.

- $\beta$  is the probability of making a Type II error

There is always a chance of making one of these errors. But, a good scientific study will minimize the chance of doing so!

Decision	Truth	
	Null Hypothesis	Alternative Hypothesis
Null Hypothesis	OK	TYPE II ERROR
Alternative Hypothesis	TYPE I ERROR	OK

## Power

The power of a statistical test is its probability of rejecting the null hypothesis if the null hypothesis is false. That is, power is the ability to correctly reject  $H_0$  and detect a significant effect. In other words, power is one minus the type II error risk.

$$\text{Power} = 1 - \beta = P(\text{reject } H_0 | H_0 \text{ is false})$$

Which error is worse?

Type I = you are innocent, yet accused of cheating on the test.

Type II = you cheated on the test, but you are found innocent.

This depends on the context of the problem too. But in most cases scientists are trying to be “conservative”; it's worse to make a spurious discovery than to fail to make a good one. Our goal is to increase the power of the test that is to minimize the length of the CI.

We need to keep in mind:

- the effect of the sample size,
- the correctness of the underlying assumptions about the population,
- statistical vs. practical significance, etc...

(see the handout). To study the tradeoffs between the sample size,  $\alpha$ , and Type II error we can use power and operating characteristic curves.

## Height Example

### One sample z-test

Assume data are independently sampled from a normal distribution with unknown mean  $\mu$  and known variance  $\sigma^2 = 9$ . Make an initial assumption that  $\mu = 65$ .

Specify the hypothesis:  $H_0: \mu = 65$   $H_A: \mu \neq 65$

z-statistic: 3.58

z-statistic follow  $N(0,1)$  distribution

One-Sample Z: Height							
Test of $\mu = 65$ vs not = 65							
The assumed standard deviation = 3							
Variable	N	Mean	StDev	SE Mean	95% CI	Z	P
Height	54	66.4630	4.3425	0.4082	(65.6628, 67.2631)	3.58	0.000

The  $p$ -value,  $< 0.0001$ , indicates that, if the average height in the population is 65 inches, it is unlikely that a sample of 54 students would have an average height of 66.4630.

Alpha = 0.05. Decision:  $p$ -value  $< \alpha$ , thus **reject the null hypothesis**.

Conclude that the average height is not equal to 65.

What type of error might we have made?

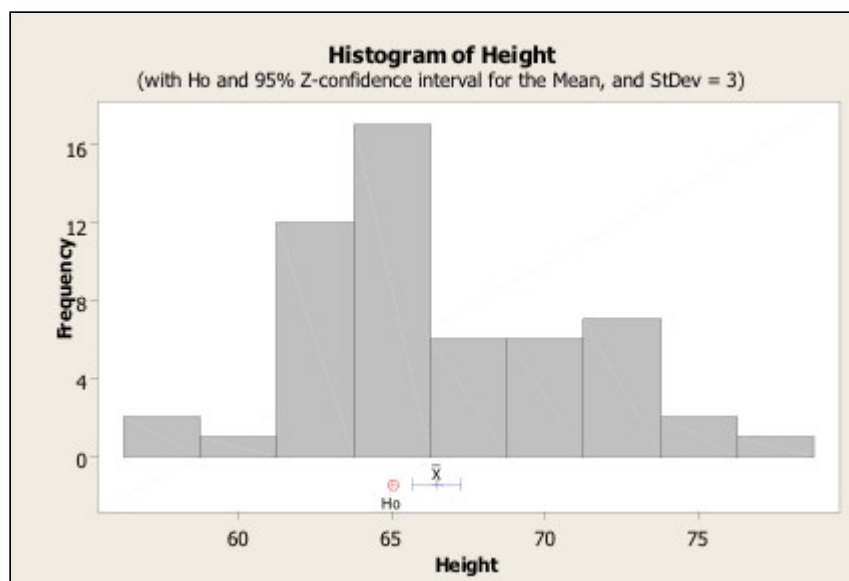
Type I error is claiming that average student height is not 65 inches, when it really is.

Type II error is failing to claim that the average student height is not 65in when it is.

We rejected the null hypothesis, i.e., claimed that the height is not 65, thus making potentially a Type I error. But sometimes the  $p$ -value is too low because of the large sample size, and we may have statistical significance but not really practical significance! That's why most statisticians are much more comfortable with using CI than tests.

## Height Example

### Graphical summary of the z-test



Based on the CI only, how do you know that you should reject the null hypothesis?

The 95% CI is (65.6628, 67.2631) ...

What about practical and statistical significance now? Is there another reason to suspect this test, and the  $p$ -value calculations?

There is a need for a further generalization. What if we can't assume that  $\sigma$  is known? In this case we would use  $s$  (the sample standard deviation) to estimate  $\sigma$ .

If the sample is very large, we can treat  $\sigma$  as known by assuming that  $\sigma = s$ . According to the law of large numbers, this is not too bad a thing to do. But if the sample is small, the fact that we have to estimate both the standard deviation and the mean adds extra uncertainty to our inference. In practice this means that we need a larger multiplier for the standard error.

We need one-sample  $t$ -test.

### One sample $t$ -test

1. Assume data are independently sampled from a normal distribution with unknown mean  $\mu$  and variance  $\sigma^2$ . Make an initial assumption,  $\mu_0$ .

2. Specify:

$H_0: \mu = \mu_0$		$H_A: \mu \neq \mu_0$
$H_0: \mu \leq \mu_0$	vs. one of those	$H_A: \mu > \mu_0$
$H_0: \mu \geq \mu_0$		$H_A: \mu < \mu_0$

3.  $t$ -statistic:  $\frac{\bar{X} - \mu_0}{s/\sqrt{n}}$  where  $s$  is a sample st.dev.

4.  $t$ -statistic follows  $t$ -distribution with  $df = n - 1$

5.  $p$ -value:

6. Alpha = 0.05, we conclude ....

### Testing for the population proportion

Let's go back to our CNN poll. Assume we have a SRS of 1,017 adults.

We are interested in testing the following hypothesis:  $H_0: p = 0.50$  vs.  $p > 0.50$

What is the test statistic?

If alpha = 0.05, what do we conclude?

We will see more details in the next lesson on proportions, then distributions, and possible tests.

Source URL: <https://onlinecourses.science.psu.edu/stat504/node/20>

## GLM Stats Model (Outcomes and Significance)

**Source:** <https://stats.idre.ucla.edu/sas/output/glm/>

This page shows an example of analysis of variance run through a general linear model (GLM) with footnotes explaining the output. The data were collected on 200 high school students, with measurements on various tests, including science, math, reading and social studies. The response variable is writing test score (**write**), from which we explore its relationship with gender (**female**) and academic program (**prog**). The model examined has the main effects of female and program type, as well as their interaction. The dataset used in this page can be downloaded from

### Class Level Information

Class	Levels	Values
female	2	0 1
prog	3	1 2 3

Number of Observations Read	200
Number of Observations Used	200

Dependent Variable: write

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	4630.36091	926.07218	13.56	<.0001
Error	194	13248.51409	68.29131		
Corrected Total	199	17878.87500			

R-Square	Coeff Var	Root MSE	write Mean
0.258985	15.65866	8.263856	52.77500

Source	DF	Type III SS	Mean Square	F Value	Pr > F
female	1	1261.853291	1261.853291	18.48	<.0001
prog	2	3274.350821	1637.175410	23.97	<.0001
female*prog	2	325.958189	162.979094	2.39	0.0946

### Class Level Information

#### Class Level Information

Class <sup>a</sup>	Levels <sup>b</sup>	Values <sup>c</sup>
female	2	0 1
prog	3	1 2 3

Number of Observations Read<sup>d</sup> 200  
 Number of Observations Used<sup>d</sup> 200

a. **Class** – Underneath are the categorical (factor) variables, which were defined as such in the **class** statement. Had the categorical variables not been defined in the **class** statement and just entered in the **model** statement, the respective variables would be treated as continuous variables, which would be inappropriate.

b. **Levels** – Underneath are the respective number of levels (categories) of the factor variables defined in the **class** statement.

c. **Values** – Underneath are the respective values of the levels for the factor variables defined in the **class** statement.

d. **Number of Observations Read** and **Number of Observations Used** – This is the number of observations read and the number of observation used in the analysis. The **Number of Observations Used** may be less than the **Number of Observations Read** if there are missing values for any variables in the equation.

### Model Information

Dependent Variable<sup>e</sup>: write

Source <sup>f</sup>	DF <sup>g</sup>	Sum of Squares <sup>h</sup>	Mean Square <sup>i</sup>	F Value <sup>j</sup>	Pr > F <sup>j</sup>
Model	5	4630.36091	926.07218	13.56	<.0001
Error	194	13248.51409	68.29131		
Corrected Total	199	17878.87500			

R-Square<sup>k</sup> 0.258985    Coeff Var<sup>l</sup> 15.65866    Root MSE<sup>m</sup> 8.263856    write Mean<sup>n</sup> 52.77500

Source <sup>o</sup>	DF <sup>p</sup>	Type III SS <sup>q</sup>	Mean Square <sup>r</sup>	F Values	Pr > F <sup>s</sup>
female	1	1261.853291	1261.853291	18.48	<.0001
prog	2	3274.350821	1637.175410	23.97	<.0001
female*prog	2	325.958189	162.979094	2.39	0.0946

e. **Dependent Variable** – This is the dependent variable in our glm model.

f. **Source** – Underneath are the sources of variation of the dependent variable. There are three parts, Model, Error, and Corrected Total. With glm, you must think in terms of the variation of the response variable (sums of squares), and partitioning this variation. The variation in the response variable, denoted by Corrected Total, can be partitioned into two unique parts. The first partition, Model, is the variance in the response accounted by our model (**female prog female\*prog**). The second source, Error, is the variation not explained by the Model. These two sources, the explained (Model), and unexplained (Error), add up to the Corrected Total,  $SS_{\text{Corrected Total}} = SS_{\text{Model}} + SS_{\text{Error}}$ .

The term “Corrected Total” is called such, as compared to “Total”, or more correctly, “Uncorrected Total,” because the “Corrected Total” adjusts the sums of squares to incorporate information on the intercept. Specifically, the Corrected Total is the sum of the squared difference between the response

variable and the mean of the response variable, whereas the Uncorrected Total is the sum of the squared values of just the response variable.

g. **DF** – These are the degrees of freedom associated with the respective sources of variance. As with the additive nature of the sums of squares, the degrees of freedom are also additive,  $DF_{Corrected\ Source} = DF_{Model} + DF_{Error}$ . The  $DF_{Corrected\ Total}$  has  $N-1$  degrees of freedom, where  $N$  is the total sample size. See DF, superscript p, for the calculation of the DF for each individual predictor variable, which add up to  $DF_{Model}$ . Hence,  $DF_{Error} = DF_{Corrected\ Total} - DF_{Model}$ . The  $DF_{Model}$  and  $DF_{Error}$  define the parameters of the F-distribution used to test F Value, superscript j.

h. **Sum of Squares** – These are the sums of squares that correspond to the three sources of variation.  $SS_{Model}$  – The Model sum of squares is the squared difference of the predicted value and the grand mean summed over all observations. Suppose our model did not explain a significant proportion of variance, then the predicted value would be near the grand mean, which would result with a small  $SS_{Model}$ , and  $SS_{Error}$  would nearly be equal to  $SS_{Corrected\ Total}$ .  $SS_{Error}$  – The Error sum of squares is the squared difference of the observed value from the predicted value summed over all observations.  $SS_{Corrected\ Total}$  – The Corrected Total sum of squares is the squared difference of the observed value from the grand mean summed over all observations.

i. **Mean Square** – These are the Mean Squares (MS) that correspond to the partitions of the total variance. The MS is defined as  $SS/DF$ .

j. **F Value** and **Pr > F** – These are the F Value and p-value, respectively, testing the null hypothesis that the Model does not explain the variance of our response variable. F Value is computed as  $MS_{Model} / MS_{Error}$ , and under the null hypothesis, F Value follows a central F-distribution with numerator DF =  $DF_{Model}$  and denominator DF =  $DF_{Error}$ . The probability of observing an F Value as large as, or larger, than 13.56 under the null hypothesis is  $< 0.0001$ . If we set our alpha level at 0.05, our willingness to accept a Type I error, we'd reject the null hypothesis and conclude that our model explains a statistically significant proportion of the variance.

k. **R-Square** – This is the R-Square value for the model. R-Square defines the proportion of the total variance explained by the Model and is calculated as  $R-Square = SS_{Model} / SS_{Corrected\ Total} = 4630.36 / 17878.88 = 0.259$ .

l. **Coeff Var** – This is the Coefficient of Variation (CV). The coefficient of variation is defined as the 100 times root MSE divided by the mean of response variable;  $CV = 100 * 8.26 / 52.775 = 15.659$ . The CV is a dimensionless quantity and allows the comparison of the variation of populations.

m. **Root MSE** – This is the root mean square error. It is the square root of the  $MS_{Error}$  and defines the standard deviation of an observation about the predicted value.

n. **write Mean** – This is the grand mean of the response variable.

o. **Source** – Underneath are the variables in the model. Our model has **female**, **prog**, and the interaction of **female** and **prog**. The interaction disallows the effect of, say, **prog**, over the levels of **female** to be additive. Also, our model follows the hierarchical principal, i.e., if an interaction term is in the model (**female\*prog**), the lower order terms (**female** and **prog**) must be included. Further, when there is a significant interaction in the model, the main effects (the lower order terms) are difficult to interpret. If the interaction term is not statistically significant, some would advise dropping the term and rerunning

the model with just the main effects, so that the main effects would have an unambiguous meaning. The traditional anova approach would leave the nonsignificant interaction in the model and interpret the main effects in the normal manner. If the interaction term is found statistically significant, one would leave the model as is and evaluate the simple main effects.

p. **DF** – These are the degrees of freedom for the individual predictor variables in the model. From the class level information section, the lower order term DF is given by the number of levels minus one. For example, **female** as two levels, therefore  $DF_{\text{female}} = 2-1=1$ . Also, **prog** has three levels and  $DF_{\text{prog}} = 3-1=2$ . For the interaction term,  $DF_{\text{female*prog}} = DF_{\text{prog}} * DF_{\text{female}} = 1*2 = 2$ . The DF of the predictor variables, along with the  $DF_{\text{Error}}$ , define the parameters of the F-distribution used to test the significance of F Value, superscript s.

q. **Type III SS** – These are the type III sum of squares, which are referred to as partial sum of squares. For a particular variable, say **female**,  $SS_{\text{female}}$  is calculated with respect to the other variables in the model, **prog** and **female\*prog**. Also, we showed earlier that  $SS_{\text{Corrected Total}} = SS_{\text{Model}} + SS_{\text{Error}}$ , and we might expect that  $SS_{\text{Model}} = SS_{\text{female}} + SS_{\text{prog}} + SS_{\text{prog*female}}$ ; however, this is generally not the case (this is only true for a balanced design).

r. **Mean Square** – These are the mean squares for the individual predictor variables in the model. They are calculated as  $SS/DF$ , and along  $MS_{\text{Error}}$ , they are used to calculate F Value, superscript s.

s. **F Value** and **Pr > F** – These are the F Value and p-value, respectively, testing the null hypothesis that an individual predictor in the model does not explain a significant proportion of the variance, given the other variables are in the model. F Value is computed as  $MS_{\text{Source Var}} / MS_{\text{Error}}$ . Under the null hypothesis, F Value follows a central F-distribution with numerator  $DF = DF_{\text{Source Var}}$ , where Source Var is the predictor variable of interest, and denominator  $DF = DF_{\text{Error}}$ . Following the point made in Source, superscript o, we focus only on the interaction term. **female\*prog** – This is the F Value and p-value testing the interaction of **female** and **prog** on the response variable, given the other variables are in the model. The probability of observing an F Value, as large as, or larger, than 2.39 under the null hypothesis that there is not an interaction of **female** and **prog**, given the other variables are in the model, is 0.0946. If we set our alpha level at 0.05, the probability of a Type I error, we would fail to reject the null hypothesis that **female** and **prog** do not interact. Based on this finding, some would advise rerunning the model without the interaction term, including only the main effects in the model (and the intercept). This would in turn permit a valid interpretation of the main effects of **female** and **prog**.

# Generalized Linear Model (GLM) Functions (ML Engine)

The GLM and GLML1L2 functions perform linear regression analysis for distribution functions using a user-specified distribution family and link function. Their output is input to the GLMPredict\_MLE and GLML1L2Predict functions (respectively), which perform generalized linear model prediction on new input data.

The GLM and GLML1L2 functions differ in these ways:

Function	Description	Supported Distribution Families	Supported Regularization Models	Output Tables
<a href="#">GLM</a>	Unbiased ordinary least square estimator	See <a href="#">Supported Family/Link Function Combinations</a>	None	<ul style="list-style-type: none"><li>Model table</li></ul>
<a href="#">GLML1L2</a>	Biased estimator based on regularization	Binomial, Gaussian	Ridge, LASSO, and elastic net	<ul style="list-style-type: none"><li>Model table</li><li>[Optional] Factor table</li></ul>

## Regularization

*Regularization* is a technique for reducing overfitting and thus decreasing the variance of trained models. GLM functions are fit by minimizing a loss function, such as the sum of squared errors. For example, given a predictor vector  $X \in \mathbb{R}^p$ , a response variable  $Y \in \mathbb{R}$ , and  $N$  observation pairs, you can find model parameters  $\beta_0$  and  $\beta$  with this formula:

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} \sum_{i=1}^N (y_i - \beta_0 - X\beta)^2$$

These fits can be regularized by adding a penalty function  $P(\beta)$  to the loss function being minimized. For example:

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} \sum_{i=1}^N (y_i - \beta_0 - X\beta)^2 + \lambda P(\beta)$$

where  $\lambda$  controls the strength of the penalty function.

For logistic regression, the loss function is based on the log likelihood, as follows:

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} - \left[ \frac{1}{N} \sum_{i=1}^N y_i \cdot (\beta_0 + x_i^T \beta) - \log(1 + e^{(\beta_0 + x_i^T \beta)}) \right] + \lambda P(\beta)$$

These are three popular penalty functions:

- The sum of the absolute values of the model parameters:

$$P(\beta) = \|\beta\|_{L1}$$

which is the L1 norm of the model parameters. This regularization technique, also called Least Absolute Shrinkage and Selection Operator (LASSO), was introduced by Robert Tibshirani in 1996. LASSO has the potential to shrink some parameters to zero; therefore, you can also use it for variable selection.

- The sum of the squared values of the model parameters:

$$P(\beta) = \|\beta\|_{L2}^2$$

which is the L2 norm of the model parameters. This regularization technique is also called *ridge regression*. With ridge regression, parameter values become smaller as  $\lambda$  increases, but never reach zero.

- Elastic net regularization, which is a linear combination of L1 and L2 normalization:

$$P_\alpha(\beta) = (1 - \alpha)\|\beta\|_{L2}^2 + \alpha\|\beta\|_{L1}$$

## References

- Friedman, J., Hastie, T., and Tibshirani, R. (2010). *Regularization Paths for Generalized Linear Models via Coordinate Descent*. *Journal of Statistical Software*, 33(1), 1 - 22. doi ([GLM regularization paths article](#))
- Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, R. J. (2012), *Strong rules for discarding predictors in lasso-type problems*. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74: 245–266. doi:10.1111/j.1467-9868.2011.01004.x

## GLM

The generalized linear model (GLM) is an extension of the linear regression model that enables the linear equation to relate to the dependent variables by a link function. The GLM function supports several distribution families and associated link functions.

You can input the output table to the function [GLMPredict\\_MLE](#).

The GLM function implementation uses the Fisher Scoring Algorithm, which scales better than the least-squares algorithm that the `glm()` function in the R package stats uses. The results of the two algorithms

usually match closely. However, when the input data is highly skewed or has a large variance, the Fisher Scoring Algorithm can diverge, and you must use data set knowledge and trial and error to select the optimal family and link functions.

If the predictors are collinear, Teradata recommends using GLML1L2 with regularization parameters.

For more information about generalized linear models, see:

- Dobson, A.J.; Barnett, A.G. (2008). *Introduction to Generalized Linear Models* (3rd ed.). Boca Raton, FL: Chapman and Hall/CRC. ISBN 1-58488-165-8.
- Hardin, James; Hilbe, Joseph (2007). *Generalized Linear Models and Extensions* (2nd ed.). College Station: Stata Press. ISBN 1-59718-014-9.

#### Related Information:

[LikelihoodRatioTest \(ML Engine\)](#)

## GLM Syntax

### GLM Syntax Elements

#### OutputTable

Specify the name for the output table of coefficients. This table must not exist.

#### TargetColumns

[Optional] Specify the name of the column that contains the dependent variable (Y) followed by the names of the columns that contain the predictor variables ( $X_i$ ), in this format: 'Y,X<sub>1</sub>,X<sub>2</sub>,...,X<sub>p</sub>'.

Default behavior: The first column of the InputTable is Y and the remaining InputTable columns are  $X_i$ , except for the column specified by the WeightColumn syntax element.

#### CategoricalColumns

[Optional] Specify columnname-value pairs, each of which contains the name of a categorical input column and the category values in that column that the function is to include in the model that it creates.

columnname_value_pair	Description
'columnname:max_cardinality'	Limits categories in column to <i>max_cardinality</i> to most common ones and groups others together as 'others'. For example, 'column_a:3' specifies that for column_a, function uses 3 most common categories and sets category of rows that do not belong to those 3 categories to 'others'.
'columnname:(category [, ...])'	Limits categories in column to those that you specify and groups others together as 'others'. For example, 'column_a : (red, yellow, blue)' specifies that for column_a, function uses categories red, yellow, and blue, and sets category of rows that do not belong to those categories to 'others'.

<i>columnname_value_pair</i>	Description
'columnname'	All category values appear in model.

If you specify the TargetColumns syntax element, the columns that you specify in the CategoricalColumns syntax element must also appear in the TargetColumns syntax element.

For information about columns that you must identify as numeric or categorical, see [Identification of Numeric and Categorical Columns](#).

### Family

[Optional] Specify the distribution exponential family, which is one of the following:

- 'BINOMIAL' (Default)
- 'LOGISTIC' (equivalent to 'BINOMIAL')
- 'POISSON'
- 'GAUSSIAN'
- 'GAMMA'
- 'INVERSE\_GAUSSIAN'
- 'NEGATIVE\_BINOMIAL'

For Binomial/Logistic and Gaussian applications with high collinearity, Teradata recommends using [GLML1L2](#) with regularization parameters instead of GLM. GLML1L2 is expected to provide better performance and accuracy.

### LinkFunction

[Optional] Specify the link function.

Default: 'CANONICAL'. The canonical link functions (default link functions) and the link functions that are allowed for each exponential family are listed in the tables in [Supported Family/Link Function Combinations](#).

### WeightColumn

[Optional] Specify the name of an InputTable column that contains the weights to assign to responses.

You can use non-NULL weights to indicate that different observations have different dispersions (with the weights being inversely proportional to the dispersions). Equivalently, when the weights are positive integers  $w_i$ , each response  $y_i$  is the mean of  $w_i$  unit-weight observations. A binomial GLM uses prior weights to give the number of trials when the response is the proportion of successes. A Poisson GLM rarely uses weights.

If the weight is less than the response value, the function throws an exception. Therefore, if the response value is greater than 1, you must specify a weight that is greater than or equal to the response value.

Default behavior: All observations have equal weight.

### StopThreshold

[Optional] Specify the convergence threshold.

Default: 0.01

### MaxIterNum

[Optional] Specify the maximum number of iterations that the algorithm runs before quitting if the convergence threshold has not been met. The parameter *max\_iterations* must be a positive INTEGER value.

Default: 25

### Intercept

[Optional] Specify whether the function uses an intercept. For example, in  $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$ , the intercept is  $\beta_0$ .

Default: 'true'

## Supported Family/Link Function Combinations

Family Name	Family Function Name	Link	Link Function Expression	Used
Binomial or Logistic	BINOMIAL or LOGISTIC	logit (default) probit cloglog log cauchit	$\log(\mu/(1-\mu))$ $\Phi^{-1}(\mu)$ $\log[-\log(1-\mu)]$ $\log(\mu)$ $\tan(\pi(\mu - 1/2))$	When the dependent variable (Y) has only two possible values (0 and 1). The algorithm applies the model to the data, predicts the most likely outcome for each input, and supplies a logit (logarithm of odds) for each outcome.
Gamma	GAMMA	inverse (default) identity log	$\mu^{-1}$ $\mu$ $\log(\mu)$	When data is continuous with constant response variance and appears to be right-skewed.
Gaussian	GAUSSIAN	identity (default) inverse log	$\mu$ $\mu^{-1}$ $\log(\mu)$	When the data is grouped around a single mean and can be graphed in a normal or bell curve distribution.
Inverse Gaussian	INVERSE_GAUSSIAN	inverse_mu_squared (default) identity inverse log	$\mu^{-2}$ $\mu$ $\mu^{-1}$ $\log(\mu)$	When the data is grouped around a single mean but the graph appears to have a right-skewed curve distribution.
Poisson	POISSON	log (default) identity square_root	$\log(\mu)$ $\mu$ $\mu^{1/2}$	To model count data (nonnegative integers) and contingency models (matrixes of the frequency distribution of variables).  The algorithm assumes that the dependent variable (Y) has a Poisson distribution (that is, that Y is segmented into intervals of, for example, time or geographic location) and then calculates the discrete probability of one or more events occurring within these segments.

Family Name	Family Function Name	Link	Link Function Expression	Used
Negative Binomial	NEGATIVE_BINOMIAL	log (default) identity	$\log(\mu)$ $\mu$	To model count data (nonnegative integers), usually over-dispersed response variables.

The following table shows the common link functions for the common distribution exponential families. D identifies the default link for each family.

Link	Link Descriptive	Binomial (Logistic)	Gamma	Gaussian	Inverse_Gaussian	Poisson	Negative_Binomial
logit	LOGIT	D					
probit	PROBIT	*					
cloglog	COMPLEMENTARY_LOG_LOG	*					
identity	IDENTITY		*	D	*	*	*
inverse	INVERSE		D	*	*		
log	LOG	*	*	*	*	D	D
$1/\mu^2$	INVERSE_MU_SQUARED				D		
sqrt	SQUARE_ROOT					*	
cauchit	CAUCHIT	*					

## GLM Input

### InputTable Schema

#### Note:

It is important to normalize the input variables before calling this function. For details, see [Normalized Input](#).

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>dependent_variable_column</i>	Any numeric SQL data type	Dependent/response variable. Cannot be NULL. Must be first in TargetColumns syntax element. If Family is BINOMIAL or LOGISTIC, each value in this column must be either 0 or 1.
<i>predictor_variable_column</i>	Any	[Column appears one or more times.] Independent/predictor variable. Cannot be NULL. Must follow <i>dependent_variable_column</i> in TargetColumns syntax element.

Column	Data Type	Description
		Teradata recommends using <a href="#">Scale</a> function on numeric predictors before calling function.
<i>categorical_column</i>	CHARACTER, VARCHAR, INTEGER, BYTEINT, DATE, TIME (without TIME ZONE)	[Column appears only with CategoricalColumns syntax element.] Categorical variable. Must also appear in TargetColumns syntax element.
<i>weight_column</i>	INTEGER, DOUBLE PRECISION	[Column appears only with WeightColumn syntax element. ] Weight to assign to response variable.

## GLM Onscreen Output

The onscreen output of the GLM function is a table containing information about the regression analysis of the data, in two sections:

- Information about the model intercept and coefficients
- Information about the regression (number of iterations and number of rows processed) and several goodness-of-fit measures

### Columns

Column	Description
predictor	Name of predictor or other reported result. For categorical predictors, the function selects one category as the reference category, and outputs one row for each other category for the column, in the format <i>predictor.level</i> . For example, if column color has categories 'red', 'blue', and 'green', and green is the reference category, the function outputs these rows: color.red color.blue
estimate	For predictors, estimated value of coefficient. For other reported results, calculated value.
std_error	For predictors, standard deviation of the mean (standard error). For other reported results, not applicable (value 0).
z_score	For predictors, calculated z-score. For other reported results, calculated value.
p_value	For predictors, calculated p-value. For other reported results, not applicable (value 0).
significance	For predictors, indicator of predictor significance. For key to significance codes, see <a href="#">CoxPH Output</a> . For other reported results, description of result.

## Rows

The onscreen output includes a row for each estimated parameter of the model and additional information about the model and the regression.

### Estimated Parameters of the Model

Parameter	Description
(Intercept)	Value of link function (Y) when all predictors are 0.
<i>predictor</i>	[Column appears only for numerical predictor.] Predictor name.
<i>predictor.level</i>	[Column appears only for categorical predictor.] Predictor name and level. Table has a row for each level of the predictor except one, which serves as the reference level.

### Model and Regression Information

Value	Description
ITERATIONS#	Number of Fisher Scoring iterations performed on function.
ROWS#	Number of rows of data received as input.
Residual deviance	Deviance, with degrees of freedom reported in significance column. Residual deviance is not displayed when Family is GAMMA, NEGATIVE_BINOMIAL, or INVERSE_GAUSSIAN
Pearson goodness of fit	Sum of squared Pearson residual.
AIC	Akaike information criterion, a measure of relative quality of model for given data set.
BIC	Bayesian information criterion, partly based on likelihood function and closely related to AIC. BIC is a criterion for model selection among a finite set of models; the model with the lowest BIC is preferred.
Wald Test	Tests goodness of fit.
Dispersion parameter	For GAUSSIAN, the value of this parameter is estimated from the data. For all other families, this parameter value is 1.

The coefficients are also stored in the table *output\_table* for later use.

For the Gamma distribution density, AIC and BIC might have the value NaN when the dispersion parameter is very small and goodness-of-fit is poor.

## GLM Output Table

The output table specified by the OutputTable syntax element stores the estimated coefficients and statistics, which are used by the functions [GLMPredict\\_MLE](#) and [LikelihoodRatioTest \(ML Engine\)](#).

## Columns

Column	Description
attribute	Numeric index of predictor.
predictor	Predictor name.
category	For categorical predictor, its level. For numeric predictor, NULL.
estimate	Estimated coefficient.
std_error	Standard error of coefficient.
t_score	[Column appears only with Family ('GAUSSIAN').] The t_score follows a $t(N-p-1)$ distribution.
z_score	[Column appears only without Family ('GAUSSIAN').] The z-score follows the $N(0,1)$ distribution.
p_value	p-value for z_score. (p-value represents significance of each coefficient.)
significance	Indicator of predictor significance. For key to symbols in this column, see <a href="#">CoxPH Output</a> .
family	Distribution exponential family, specified by Family syntax element.

## Rows

The OutputTable includes a row for each of the following parameters.

Parameter	Description
Loglik	Log likelihood of model.
(Intercept)	Value of link function (Y) when all predictors are 0.
Predictors	Predictor name. For categorical predictor, table has a row for each level of the predictor.

## Odds Ratio and Confidence Intervals

You can exponentiate the coefficients (the estimate column in the output table coefficient estimates) and interpret them as odds ratios (ORs). To perform this type of computation, you can run the following SQL queries on the output of the GLM function.

```
-- odds ratios only
SELECT predictor, category,
       EXP(estimate) AS odds_ratio
FROM glm_output;
-- odds ratios and 95% CI
SELECT predictor, category,
       EXP(estimate) AS odds_ratio,
```

```
EXP(estimate - 1.96 * std_err) AS lower_bound,
EXP(estimate + 1.96 * std_err) AS upper_bound
FROM glm_output;
```

## Goodness-of-Fit Tests

- [Deviance](#)
- [Wald Test](#)
- [Pearson's Chi-squared Statistic](#)

### Deviance

The deviance for a model  $M_0$ , based on a data set  $y$ , is defined as follows:

$$D(y) = -2(\log(p(y | \hat{\theta}_0)) - \log(p(y | \hat{\theta}_s)))$$

In the preceding equation:

$$\hat{\theta}_0$$

denotes the fitted values of the parameters in the model  $M_0$ .

$$\hat{\theta}_s$$

denotes the fitted parameters for the full model (or saturated model).

Both sets of fitted values are implicitly functions of the observations  $y$ . In this case, the full model is a model with a parameter for every observation so that the data are fitted exactly.

The deviance is used to compare two models—in particular in the case of generalized linear models where it has a similar role to residual variance from ANOVA in linear models (RSS).

Suppose in the framework of the GLM that there are two nested models,  $M_1$  and  $M_2$ . In particular, suppose that  $M_1$  contains the parameters in  $M_2$ , and  $k$  additional parameters. Then, under the null hypothesis that  $M_2$  is the true model, the difference between the deviances for the two models follows an approximate chi-squared distribution with  $k$ -degrees of freedom. This provides us an alternative way for computing the log-likelihood ratio of two models.

Deviance is implemented in the GLM function. It computes residual deviance from model deviance and saturated deviance. The function does not compute null deviance.

## Wald Test

Significance tests can be performed for individual regression coefficients (that is,  $H_0 : \beta_j = 0$ ) by computing the Wald statistics, which are similar to the partial t-statistics from classical regression:

$$w_j = \frac{\hat{\beta}_j}{\hat{\text{se}}(\hat{\beta}_j)}$$

Under the null hypothesis that  $\beta_j = 0$ , the Wald test statistic  $w_j$  follows approximately a standard normal distribution (and its square is approximately a chi-square on one-degree of freedom).

This quantity is computed by the GLM function as the Wald Test, as well as the corresponding 'p\_value'. It is in the output table, and also displayed on the screen.

## Pearson's Chi-squared Statistic

The deviance generalizes the sum of squared errors. Another generalization of sum of squared errors is Pearson's chi-squared statistic. Given a generalized linear model with responses  $y_i$ , weights  $w_i$ , fitted means  $\mu_i$ , variance function  $v(\mu)$  and dispersion  $\phi = 1$ , the Pearson goodness-of-fit statistic is

$$X^2 = \sum \frac{w_i(y_i - \hat{\mu}_i)^2}{v(\hat{\mu}_i)}$$

If the fitted model is correct and the observations  $y_i$  are approximately normal,  $X^2$  is approximately distributed as  $X^2$  on the residual degrees of freedom for the model. Both the deviance and the generalized Pearson  $X^2$  have exact  $X^2$  distributions for Normal-theory linear models (assuming of course that the model is true), and asymptotic results are available for the other distributions. The deviance has a general advantage as a measure of discrepancy in that it is additive for nested sets of models if maximum-likelihood estimates are used, whereas  $X^2$  in general is not. However,  $X^2$  may sometimes be preferred because of its more direct interpretation.

The GLM function computes the Pearson's goodness of fit.

## GLM Examples

### GLM Example: Logistic Regression Analysis with Intercept

In logistic regression, the dependent variable (Y) has only two possible values (0 and 1, 'yes' and 'no', or 'true' and 'false'). The algorithm applies the model to the data and predicts the most likely outcome.

## Input

The InputTable, `admissions_train`, contains data about applicants to an academic program. For each applicant, attributes in the table include a Masters Degree indicator, a grade point average (on a 4.0 scale), a statistical skills indicator, a programming skills indicator, and an indicator of whether the applicant was admitted. The Masters Degree, statistical skills, and programming skills indicators are categorical variables. Masters degree has two categories (yes or no), while the other two have three categories (Novice, Beginner and Advanced). For admitted status, 1 indicates that the student was admitted and 0 indicates otherwise.

**InputTable: admissions\_train**

id	masters	gpa	stats	programming	admitted
1	yes	3.95	Beginner	Beginner	0
2	yes	3.76	Beginner	Beginner	0
3	no	3.7	Novice	Beginner	1
4	yes	3.5	Beginner	Novice	1
5	no	3.44	Novice	Novice	0
6	yes	3.5	Beginner	Advanced	1
7	yes	2.33	Novice	Novice	1
8	no	3.6	Beginner	Advanced	1
9	no	3.82	Advanced	Advanced	1
10	no	3.71	Advanced	Advanced	1
11	no	3.13	Advanced	Advanced	1
12	no	3.65	Novice	Novice	1
13	no	4	Advanced	Novice	1
14	yes	3.45	Advanced	Advanced	0
15	yes	4	Advanced	Advanced	1
16	no	3.7	Advanced	Advanced	1
17	no	3.83	Advanced	Advanced	1
18	yes	3.81	Advanced	Advanced	1
19	yes	1.98	Advanced	Advanced	0
20	yes	3.9	Advanced	Advanced	1
21	no	3.87	Novice	Beginner	1
22	yes	3.46	Novice	Beginner	0

id	masters	gpa	stats	programming	admitted
23	yes	3.59	Advanced	Novice	1
24	no	1.87	Advanced	Novice	1
25	no	3.96	Advanced	Advanced	1
26	yes	3.57	Advanced	Advanced	1
27	yes	3.96	Advanced	Advanced	0
28	no	3.93	Advanced	Advanced	1
29	yes	4	Novice	Beginner	0
30	yes	3.79	Advanced	Novice	0
31	yes	3.5	Advanced	Beginner	1
32	yes	3.46	Advanced	Beginner	0
33	no	3.55	Novice	Novice	1
34	yes	3.85	Advanced	Beginner	0
35	no	3.68	Novice	Beginner	1
36	no	3	Advanced	Novice	0
37	no	3.52	Novice	Novice	1
38	yes	2.65	Advanced	Beginner	1
39	yes	3.75	Advanced	Beginner	0
40	yes	3.95	Novice	Beginner	0

## SQL Call

The response variable (admitted, in this example) must be specified as the first variable listed in the TargetColumns syntax element, followed by the other predictors.

```
DROP TABLE glm_admissions_model;

SELECT * FROM GLM (
  ON admissions_train AS InputTable
  OUT TABLE OutputTable (glm_admissions_model)
  USING
  TargetColumns ('admitted','masters', 'gpa', 'stats', 'programming')
  CategoricalColumns ('masters', 'stats', 'programming')
  Family ('LOGISTIC')
  LinkFunction ('LOGIT')
```

```

WeightColumn ('1')
StopThreshold (0.01)
MaxIterNum (25)
Intercept ('true')
) AS dt;

```

## Output

The output table shows the model statistics.

predictor	estimate	std_error	z_score
p_value	significance		
-----			
-----			
-----			
(Intercept)	1.0775099992752075	2.920759916305542	
0.36891400814056396	0.7121919989585876		
masters.no	2.21655011177063	1.0199899673461914	
2.173110008239746	0.029771899804472923 *		
gpa	-0.11393500119447708	0.802573025226593	
-0.14196200668811798	0.8871099948883057		
stats.novice	0.04068480059504509	1.1156699657440186	
0.036466699093580246	0.9709100127220154		
stats.beginner	0.5266180038452148	1.2229000329971313	
0.43063101172447205	0.6667360067367554		
programming.beginner	-1.769760012626648	1.069000005722046	
-1.6555299758911133	0.09781769663095474 .		
programming.novice	-0.9803500175476074	1.1400400400161743	
-0.8599230051040649	0.389831006526947		
ITERATIONS #	4.0	0.0	
0.0	0.0 Number of Fisher Scoring iterations		
ROWS #	40.0	0.0	
0.0	0.0 Number of rows		
Residual deviance	38.90380096435547	0.0	
0.0	0.0 on 33 degrees of freedom		
Pearson goodness of fit	37.79050064086914	0.0	
0.0	0.0 on 33 degrees of freedom		
AIC	52.90380096435547	0.0	
0.0	0.0 Akaike information criterion		
BIC	64.72595977783203	0.0	
0.0	0.0 Bayesian information criterion		
Wald Test	9.896419525146484	0.0	
0.0 0.19451963901519775			
Dispersion parameter	1.0	0.0	
0.0	0.0 Taken to be 1 for BINOMIAL and POISSON.		

For categorical variables, the model selects a reference category. This example uses the Advanced category as a reference for the stats variable.

This query returns the following table:

```
SELECT * FROM glm_admissions_model;
```

attribute predictor	category	estimate	std_err
z_score	p_value	significance	family
-----	-----	-----	-----
-1 Loglik	NULL	-19.451900482177734	40.0
6.0	0.0 NULL	LOGISTIC	
0 (Intercept)	NULL	1.0775099992752075	2.920759916305542
0.36891400814056396	0.7121919989585876	LOGISTIC	
1 masters	yes	NULL	NULL
NULL	NULL NULL	LOGISTIC	
2 masters	no	2.21655011177063	1.0199899673461914
2.173110008239746	0.029771899804472923 *	LOGISTIC	
3 gpa	NULL	-0.11393500119447708	0.802573025226593
-0.14196200668811798	0.8871099948883057	LOGISTIC	
4 stats	advanced	NULL	NULL
NULL	NULL NULL	LOGISTIC	
5 stats	novice	0.04068480059504509	1.1156699657440186
0.036466699093580246	0.9709100127220154	LOGISTIC	
6 stats	beginner	0.5266180038452148	1.2229000329971313
0.43063101172447205	0.6667360067367554	LOGISTIC	
7 programming	advanced	NULL	NULL
NULL	NULL NULL	LOGISTIC	
8 programming	beginner	-1.769760012626648	1.069000005722046
-1.6555299758911133	0.09781769663095474 .	LOGISTIC	
9 programming	novice	-0.9803500175476074	1.1400400400161743
-0.8599230051040649	0.389831006526947	LOGISTIC	

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLM Example: Gaussian Distribution Analysis

For the Gaussian distribution the response variable must be a continuous numerical variable, where the data is grouped around a single mean and the graph looks like a normal or bell curve distribution. This example uses default options.

**Input**

- InputTable: housing\_train, as in [DecisionForest Example: TreeType \('classification'\), OutOfBag \('false'\)](#)

**SQL Call**

The family is GAUSSIAN and the default family link is IDENTITY.

```
DROP TABLE glm_housing_model;

SELECT * FROM GLM (
  ON housing_train AS InputTable
  OUT TABLE OutputTable (glm_housing_model)
  USING
  TargetColumns ('price', 'lotsize', 'bedrooms', 'bathrms',
                 'stories', 'garagepl', 'driveway', 'recroom',
                 'fullbase', 'gashw', 'airco', 'prefarea', 'homestyle')
  CategoricalColumns ('driveway', 'recroom', 'fullbase', 'gashw',
                     'airco', 'prefarea', 'homestyle')
  Family ('GAUSSIAN')
  LinkFunction ('IDENTITY')
  WeightColumn ('1')
  StopThreshold (0.01)
  MaxIterNum (25)
  Intercept ('true')
) AS dt;
```

**Output**

predictor	estimate	std_error	t_score
p_value	significance		
-----			
-----			
-----			
(Intercept)	36349.30078125	2733.462158203125	
13.297897338867188	0.0 ***		
lotsize	2.0809500217437744	0.26133036613464355	7.962909698486328
1.2434497875801753E-14 ***			
bedrooms	782.093017578125	766.8397216796875	
1.0198911428451538	0.3082960247993469		
bathrms	6772.31005859375	1106.7789306640625	
6.118936538696289	1.963181173181283E-9 ***		
stories	2445.6201171875	694.1449584960938	
3.523212194442749	4.673068760894239E-4 ***		

```

garagepl          1483.0999755859375    623.5965576171875
2.378300666809082  0.01778467558324337 *
driveway.no       -2822.6298828125    1481.2474365234375
-1.905576229095459  0.05730487406253815 .
recroom.yes       1208.530029296875    1358.570556640625
0.8895599842071533  0.3741496801376343
fullbase.yes      3588.300048828125    1167.3746337890625
3.0738205909729004  0.0022341914009302855 **
gashw.yes         5787.25    2405.470458984375
2.4058704376220703  0.01651271991431713 *
airco.yes         6478.7900390625    1152.1597900390625
5.623169422149658  3.1934060729099656E-8 ***
prefarea.yes      6465.64013671875    1212.8397216796875
5.330992698669434  1.5088656368789088E-7 ***
homestyle.classic -16550.900390625    1308.585205078125
-12.647933959960938  0.0 ***
homestyle.bungalow 37577.69921875    1850.173828125
20.310361862182617  0.0 ***
ITERATIONS #      2.0    0.0
0.0    0.0 Number of Fisher Scoring iterations
ROWS #            492.0    0.0
0.0    0.0 Number of rows
Residual deviance  Infinity    0.0
0.0    0.0 on 478 degrees of freedom
Pearson goodness of fit  5.3066899456E10    0.0
0.0    0.0 on 478 degrees of freedom
AIC               Infinity    0.0
0.0    0.0 Akaike information criterion
BIC               Infinity    0.0
0.0    0.0 Bayesian information criterion
Wald Test         23174.041015625    0.0
0.0    0.0 ***
Dispersion parameter  1.11018616E8    0.0
0.0    0.0 Taken to be 1 for BINOMIAL and POISSON.

```

Many predictors are significant at 95% confidence level (p-value < 0.05).

```
SELECT * FROM glm_housing_model ORDER BY attribute;
```

```

attribute predictor  category estimate          std_err
z_score             p_value          significance family
-----
-1 Loglik          NULL          -Infinity          492.0
13.0               0.0 NULL          GAUSSIAN

```

0 (Intercept)	NULL	36349.30078125	2733.462158203125
13.297897338867188		0.0 ***	GAUSSIAN
1 lotsize	NULL	2.0809500217437744	0.26133036613464355
7.962909698486328	1.2434497875801753E-14 ***		GAUSSIAN
2 bedrooms	NULL	782.093017578125	766.8397216796875
1.0198911428451538	0.3082960247993469		GAUSSIAN
3 bathrms	NULL	6772.31005859375	1106.7789306640625
6.118936538696289	1.963181173181283E-9 ***		GAUSSIAN
4 stories	NULL	2445.6201171875	694.1449584960938
3.523212194442749	4.673068760894239E-4 ***		GAUSSIAN
5 garagepl	NULL	1483.0999755859375	623.5965576171875
2.378300666809082	0.01778467558324337 *		GAUSSIAN
6 driveway	yes	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
7 driveway	no	-2822.6298828125	1481.2474365234375
-1.905576229095459	0.05730487406253815 .		GAUSSIAN
8 recroom	no	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
9 recroom	yes	1208.530029296875	1358.570556640625
0.8895599842071533	0.3741496801376343		GAUSSIAN
10 fullbase	no	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
11 fullbase	yes	3588.300048828125	1167.3746337890625
3.0738205909729004	0.0022341914009302855 **		GAUSSIAN
12 gashw	no	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
13 gashw	yes	5787.25	2405.470458984375
2.4058704376220703	0.01651271991431713 *		GAUSSIAN
14 airco	no	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
15 airco	yes	6478.7900390625	1152.1597900390625
5.623169422149658	3.1934060729099656E-8 ***		GAUSSIAN
16 prefarea	no	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
17 prefarea	yes	6465.64013671875	1212.8397216796875
5.330992698669434	1.5088656368789088E-7 ***		GAUSSIAN
18 homestyle	eclectic	NULL	NULL
NULL	NULL NULL	GAUSSIAN	
19 homestyle	classic	-16550.900390625	1308.585205078125
-12.647933959960938	0.0 ***		GAUSSIAN
20 homestyle	bungalow	37577.69921875	1850.173828125
20.310361862182617	0.0 ***		GAUSSIAN

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLMPredict\_MLE

The GLMPredict\_MLE function uses the model output by the function [GLM](#) to perform generalized linear model prediction on new input data.

## GLMPredict\_MLE Syntax

### Version 1.15

```
SELECT * FROM GLMPredict_MLE (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION [ ORDER BY attribute, predictor ]
  [ USING
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
    [ Family ('family') ]
    [ LinkFunction ('link') ]
    [ OutputProb ({'true'|'t' | 'yes'|'y' | '1' | 'false'|'f' | 'no'|'n' | '0'}) ]
    [ Responses ('response' [,...]) ]
  ]
) AS alias;
```

### Related Information:

[Comments in Queries](#)

[Column Specification Syntax Elements](#)

## GLMPredict\_MLE Syntax Elements

### Accumulate

[Optional] Specify the names of input table columns to copy to the output table.

### Family

[Optional] Specify the distribution exponential family.

If you specify this syntax element, you must give it the same value that you used for the Family syntax element of the function [GLM](#) when you created the Model table.

Default: Read from the Model table

### LinkFunction

[Optional] Specify the link function. For the canonical link functions (default link functions) and the link functions allowed for each exponential family, see [Supported Family/Link Function Combinations](#).

If you specify this syntax element, you must give it the same value that you used for the LinkFunction syntax element of the function [GLM](#) when you created the Model table.

Default: 'CANONICAL'

### OutputProb

[Family must be BINOMIAL or LOGISTIC. Required with Responses, optional otherwise.]

Specify whether to output the probability for each *response*. If you omit Responses, the function outputs the probability of the predicted response.

Default: 'false'

### Responses

[Optional] Specify the labels for which to output probabilities. A label (*response*) must be 0 or 1.

Default behavior: The function outputs only the probability of the predicted class.

## GLMPredict\_MLE Input

Table	Description
Input	Contains new data.
Model	<a href="#">GLM Output Table</a>

### Input Table Schema

#### Note:

It is important to normalize the input variables before calling this function. For details, see [Normalized Input](#).

Column	Data Type	Description
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.
<i>dependent_variable_column</i>	Any	Dependent/response variables. Cannot be NULL.
<i>predictor_variable_column</i>	Any	[Column appears one or more times.] Independent/predictor variable. Cannot be NULL.

## GLMPredict\_MLE Output

### Output Table Schema

Column	Data Type	Description						
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.						
fitted_value	DOUBLE PRECISION	Score of the input data, given by equation $g^{-1}(X\beta)$ , where $g^{-1}$ is the inverse link function, X the predictors, and $\beta$ is the vector of coefficients estimated by the GLM function. For BINOMIAL classification, a predicted value close to 1 indicates a high probability of class 1. A predicted value close to 0 indicates a high probability of class 0. For other values of Family, the scores are the expected values of dependent/response variable, conditional on the predictors.						
prediction	INTEGER	[Column appears only with Family ('BINOMIAL') or Family ('LOGISTIC').] Predicted value (0 or 1).						
prob	DOUBLE PRECISION	[Column appears only if you specify OutputProb ('true') and omit Responses.] Probability that observation belongs to predicted class, calculated as follows: <table><tr><th>prediction</th><th>prob</th></tr><tr><td>0</td><td>1 - fitted_value</td></tr><tr><td>1</td><td>fitted_value</td></tr></table>	prediction	prob	0	1 - fitted_value	1	fitted_value
prediction	prob							
0	1 - fitted_value							
1	fitted_value							
prob_0	DOUBLE PRECISION	[Only with Responses, one column appears for each <i>response</i> .] Probability that observation belongs to category 0, which is 1 - fitted_value.						
prob_1	DOUBLE PRECISION	[Only with Responses, one column appears for each <i>response</i> .] Probability that observation belongs to category 1, which is fitted_value.						

## GLMPredict\_MLE Examples

### GLMPredict\_MLE Example: Logistic Distribution Prediction

#### Input

- Input: admissions\_test, which has admissions information for 20 students
- Model: glm\_admissions\_model, output by [GLM Example: Logistic Regression Analysis with Intercept](#)

admissions\_test

id	masters	gpa	stats	programming	admitted
50	yes	3.95	Beginner	Beginner	0
51	yes	3.76	Beginner	Beginner	0
52	no	3.7	Novice	Beginner	1
53	yes	3.5	Beginner	Novice	1
54	yes	3.5	Beginner	Advanced	1
55	no	3.6	Beginner	Advanced	1
56	no	3.82	Advanced	Advanced	1
57	no	3.71	Advanced	Advanced	1
58	no	3.13	Advanced	Advanced	1
59	no	3.65	Novice	Novice	1
60	no	4	Advanced	Novice	1
61	yes	4	Advanced	Advanced	1
62	no	3.7	Advanced	Advanced	1
63	no	3.83	Advanced	Advanced	1
64	yes	3.81	Advanced	Advanced	1
65	yes	3.9	Advanced	Advanced	1
66	no	3.87	Novice	Beginner	1
67	yes	3.46	Novice	Beginner	0
68	no	1.87	Advanced	Novice	1
69	no	3.96	Advanced	Advanced	1

**SQL Call**

```

CREATE MULTISET TABLE glmpredict_admissions AS (
  SELECT * FROM GLMPredict_MLE (
    ON admissions_test PARTITION BY ANY
    ON glm_admissions_model AS Model DIMENSION
    USING
    Accumulate ('id', 'masters', 'gpa', 'stats', 'programming', 'admitted')
    Family ('LOGISTIC')
    LinkFunction ('LOGIT')
    OutputProb ('t')
  )

```

```
) AS dt
) WITH DATA;
```

## Output

```
SELECT * FROM glmpredict_admissions ORDER BY 1;
```

id	masters	gpa	stats	programming	admitted	fitted_value	prediction
50	yes	3.95	beginner	beginner	0	0.3507656829365149	0
51	yes	3.76	beginner	beginner	0	0.3557112671780229	0
52	no	3.7	novice	beginner	1	0.7583079903427496	1
53	yes	3.5	beginner	novice	1	0.5560152436940663	1
54	yes	3.5	beginner	advanced	1	0.7694761266019933	1
55	no	3.6	beginner	advanced	1	0.9680314543695169	1
56	no	3.82	advanced	advanced	1	0.9457732442937538	1
57	no	3.71	advanced	advanced	1	0.9464124273756033	1
58	no	3.13	advanced	advanced	1	0.949666669516694	1
59	no	3.65	novice	novice	1	0.8741907950304955	1
60	no	4.0	advanced	novice	1	0.8650601698708184	1
61	yes	4.0	advanced	advanced	1	0.6506209990774642	1
62	no	3.7	advanced	advanced	1	0.9464701812067841	1
63	no	3.83	advanced	advanced	1	0.9457147816580886	1
64	yes	3.81	advanced	advanced	1	0.6555256147282206	1
65	yes	3.9	advanced	advanced	1	0.6532064285302687	1
66	no	3.87	novice	beginner	1	0.7547403697792542	1

```

67 yes      3.46 novice  beginner      0 0.2600362186838019      0
0.7399637813161981
68 no       1.87 advanced novice      1 0.8909664987530864      1
0.8909664987530864
69 no       3.96 advanced advanced      1 0.9449493421287761      1
0.9449493421287761

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLMPredict\_MLE Example: Gaussian Distribution Prediction

The example uses the Gaussian model created in [GLM Example: Gaussian Distribution Analysis](#) to predict house prices. To evaluate the accuracy of the model, the example calculates the root mean square error (RMSE) between the known actual price from the input table and the price predicted by the model.

### Input

- Input table: housing\_test, which contains test data for 54 houses
- Model: glm\_housing\_model from the output section of [GLM Example: Gaussian Distribution Analysis](#)

housing\_test

sn	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	g
13	27000	1700	3	1	2	yes	no	no	no	no	0
16	37900	3185	2	1	1	yes	no	no	no	yes	0
25	42000	4960	2	1	1	yes	no	no	no	no	0
38	67000	5170	3	1	4	yes	no	no	no	yes	0
53	68000	9166	2	1	1	yes	no	yes	no	yes	2
104	132000	3500	4	2	2	yes	no	no	yes	no	2
111	43000	5076	3	1	1	no	no	no	no	no	0
117	93000	3760	3	1	2	yes	no	no	yes	no	2
132	44500	3850	3	1	2	yes	no	no	no	no	0
140	43000	3750	3	1	2	yes	no	no	no	no	0
142	40000	2650	3	1	2	yes	no	yes	no	no	1
157	60000	2953	3	1	2	yes	no	yes	no	yes	0
161	63900	3162	3	1	2	yes	no	no	no	yes	1

sn	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	g
...	...	...	...	...	...	...	...	...	...	...	...

## SQL Call

The canonical link specifies the default family link, which is "identity" for the Gaussian distribution.

```
CREATE MULTISET TABLE glmpredict_housing AS (
  SELECT * FROM GLMPredict_MLE (
    ON housing_test PARTITION BY ANY
    ON glm_housing_model AS Model DIMENSION
    USING
    Accumulate ('sn', 'price')
    Family ('GAUSSIAN')
    LinkFunction ('CANONICAL')
  ) AS dt
) WITH DATA;
```

## Output

The fitted\_value column gives the predicted home price.

```
SELECT * FROM glmpredict_housing ORDER BY 1;
```

sn	price	fitted_value
---	-----	-----
13	27000.0	37345.84477329254
16	37900.0	43687.13245987892
25	42000.0	40902.02870941162
38	67000.0	72487.67201280594
53	68000.0	79238.69493055344
104	132000.0	111528.00744915009
111	43000.0	39102.882046699524
117	93000.0	66936.95215988159
132	44500.0	41819.88732004166
140	43000.0	41611.79231786728
142	40000.0	44394.14731836319
157	60000.0	66571.26562905312
161	63900.0	64900.98411035538
162	130000.0	107759.1224937439
176	57500.0	73438.73871564865
177	70000.0	62378.35326194763
195	33000.0	37197.9376707077
198	40500.0	47308.08242368698
224	78500.0	67232.86958527565

```

234 32500.0 35237.16528749466
237 43000.0 46593.47125959396
239 26000.0 43377.86666679382
249 44500.0 37863.84167766571
251 48500.0 45096.38719415665
254 60000.0 74665.70030021667
255 61000.0 60214.16523933411
260 41000.0 43066.21673202515
274 64900.0 67232.45221805573
294 47000.0 38987.55468940735
301 55000.0 55621.6930809021
306 64000.0 67339.69788479805
317 80000.0 65655.120470047
329 115442.0 123612.01978111267
339 141000.0 126282.07774448395
340 62500.0 58474.835211753845
353 78500.0 67485.69113445282
355 86900.0 68425.80433177948
364 72000.0 77422.12543773651
367 114000.0 128556.01284217834
377 140000.0 127201.90244436264
401 92500.0 84040.80987596512
403 77500.0 79857.25416207314
408 87500.0 76218.38956928253
411 90000.0 78179.1003665924
440 69000.0 80549.23930311203
441 51900.0 64670.294174194336
443 65000.0 61704.09422302246
459 44555.0 42818.367908000946
463 49000.0 49293.44947862625
469 55000.0 61779.35452270508
472 60500.0 63767.0579059124
527 105000.0 119762.26224088669
530 108000.0 116119.24969100952
540 85000.0 73146.08736228943

```

**RMSE**

This query returns the following table:

```

SELECT SQRT(AVG(POWER(glmpredict_housing.price -
  glmpredict_housing.fitted_value, 2))) AS RMSE FROM glmpredict_housing;

```

RMSE

-----  
10246.752127962065

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLML1L2

The GLML1L2 function differs from the [GLM](#) function in these ways:

- GLML1L2 supports the regularization models ridge, LASSO, and elastic net.
- GLML1L2 outputs a model table and, optionally, a factor table (GLM outputs only a model table).

You can input the model table and factor table to the [GLML1L2Predict](#) function.

## GLML1L2 Syntax

### Version 1.17

```
SELECT * FROM GLML1L2 (
  ON { table | view | (query) } AS InputTable
  [ OUT TABLE FactorTable (factor_table) ]
  USING
  TargetColumns ('target_column' [...])
  [ CategoricalColumns (({ 'categorical_column' [...]) ) ]
  ResponseColumn ('response_column')
  [ Family ({ 'BINOMIAL' | 'GAUSSIAN' }) ]
  [ Alpha (alpha) ]
  [ RegularizationLambda (lambda) ]
  [ StopThreshold (threshold) ]
  [ MaxIterNum (max_iterations) ]
) AS alias;
```

## GLML1L2 Syntax Elements

### FactorTable

[Optional] Specify the name for the FactorTable. The function encodes categorical predictors as integer values in the FactorTable and copies numeric predictors to the FactorTable unchanged.

You must also specify CategoricalColumns.

You can use *factor\_table* as InputTable for future GLML1L2 function calls, thereby saving the function from repeating the categorical-to-numerical conversion.

**TargetColumns**

Specify the names of the InputTable columns that contain the variables to use as predictors (independent variables) in the model.

**CategoricalColumns**

[Optional] Specify the names of the InputTable columns to treat as categorical variables, and which of their categories to use in the model.

<i>categorical_column_and_categories</i>	Descriptions
' <i>categorical_column</i> : <i>max_cardinality</i> '	Uses most common categories in <i>categorical_column</i> and groups other categories into category 'others'. For example, 'column_a:3' specifies that for column_a, function uses 3 most common categories and sets category of rows that do not belong to those 3 categories to 'others'.
' <i>categorical_column</i> : ( <i>category</i> [, ...])'	Uses specified categories of <i>categorical_column</i> and groups other categories into category 'others'. For example, 'column_a : (red, yellow, blue)' specifies that for column_a, function uses categories red, yellow, and blue, and sets category of rows that do not belong to those categories to 'others'.
' <i>categorical_column</i> '	Uses all categories in <i>categorical_column</i> .

If you use this syntax element, you must also specify the FactorTable syntax element, and in the TargetColumns syntax element, you must specify each *categorical\_column*.

For information about columns that you must identify as numeric or categorical, see [Identification of Numeric and Categorical Columns](#).

Default behavior: The function treats all variables as numerical.

**ResponseColumn**

Specify the name of the InputTable column that contains the responses.

**Family**

[Optional] Specify the distribution exponential family.

Default: 'GAUSSIAN'

**Alpha**

[Optional] Specify the mixing parameter for penalty computation (see the following table). The *alpha* must be in [0, 1]. If *alpha* is in (0, 1), it represents  $\alpha$  in the elastic net regularization formula in [Generalized Linear Model \(GLM\) Functions \(ML Engine\)](#).

<i>alpha</i>	Regularization Type	Parameter Description
0	Ridge	$\frac{1}{2} \ \beta\ _{L_2}^2$

<i>alpha</i>	Regularization Type	Parameter Description
(0,1)	Elastic net	$\left(\frac{(1 - \alpha)}{2}\right) \ \beta\ _{L2}^2 + \alpha \ \beta\ _{L1}$
1	LASSO	$\ \beta\ _{L1}$

Default: 0

### RegularizationLambda

[Optional] Specify the parameter that controls the magnitude of the regularization term. The value *lambda* must be in the range [0, 100]. The value 0 disables regularization.

Default: 0

### StopThreshold

[Optional] Specify the convergence threshold. The *threshold* must be a nonnegative DOUBLE PRECISION value.

Default: 1.0e<sup>-7</sup>

### MaxIterNum

[Optional] Specify the maximum number of iterations over the data. The parameter *max\_iterations* must be a positive INTEGER value in the range [1, 100000].

Default: 10000

## GLML1L2 Input

### InputTable Schema

#### Note:

It is important to normalize the input variables before calling this function. For details, see [Normalized Input](#).

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>response_column</i>	Any numeric SQL data type	Dependent/response variable. <b>Tip:</b> Remove NULL values to optimize function execution time.
<i>target_column</i>	Any	[Column appears one or more times.] Independent/predictor variable.

Column	Data Type	Description
		<b>Tip:</b> <ul style="list-style-type: none"> <li>Remove NULL values to optimize function execution time.</li> <li>Use <a href="#">Scale</a> function on numeric predictors before calling the function.</li> </ul>
<i>categorical_column</i>	CHARACTER, VARCHAR, INTEGER, BYTEINT, DATE, TIME	[Column appears once for each categorical <i>target_column</i> .] Categorical independent/predictor variable. Variable name cannot be a Teradata reserved keyword, start with a digit, or contain any nonalphanumeric character except underscore (_). Do not enclose variable name in single quotation marks.

## GLML1L2 Output

### Output Table Schema

The function displays the output table to the screen.

Column	Data Type	Description
attribute	VARCHAR	Name of model attribute.
category	VARCHAR	[Column appears only for categorical predictor.] Category (level) of predictor.
estimate	DOUBLE PRECISION	Estimate of model coefficient.
information	VARCHAR	Value of nonnumeric attribute, followed by "p" if predictor is used in model.

### FactorTable Schema

Each row in the factor table corresponds to a row in the input table.

Column	Data Type	Description
<i>target_column</i>	Categorical column: INTEGER Numeric column: Same as in InputTable	Categorical column: dummy variable Numeric column: Same as in InputTable.
<i>response_column</i>	DOUBLE PRECISION	Column copied from InputTable.

## GLML1L2 Examples

### GLML1L2 Example: Ridge Regression, Family ('BINOMIAL')

#### Input

The input table is admission\_train, as in [GLM Example: Logistic Regression Analysis with Intercept](#).

#### SQL Call

Because the response variable is binary (the admitted column has two possible values), the call specifies Family ('BINOMIAL'). Alpha (0) indicates L2 (ridge regression) regularization.

```
DROP TABLE admissions_model;
DROP TABLE admissions_factor_table;

CREATE MULTiset TABLE admissions_model AS (
  SELECT * FROM GLML1L2 (
    ON admissions_train AS InputTable
    OUT TABLE FactorTable (admissions_factor_table)
    USING
    TargetColumns ('masters', 'gpa', 'stats', 'programming')
    CategoricalColumns ('masters', 'stats', 'programming')
    ResponseColumn ('admitted')
    Family ('BINOMIAL')
    Alpha (0)
    RegularizationLambda (0.02)
  ) AS dt
) WITH DATA;
```

#### Output

```
SELECT * FROM admissions_model;
```

attribute	category	estimate	information
-----	-----	-----	-----
AIC	NULL	15.21927981934978	NULL
programming	beginner	-1.0259430213730834	p
Features #	NULL	6.0	NULL
programming	novice	-0.0820786516340258	p
masters	yes	-1.2652530272653697	p
Iterations #	NULL	28.0	NULL
Lambda	NULL	0.02	NULL
Alpha	NULL	0.0	NULL
stats	beginner	0.08063465501463249	p

```

Regularization NULL NULL Ridge
stats novice -0.026716553307241597 p
Family NULL NULL Binomial
Converged NULL NULL true
gpa NULL 0.38346423433872745 p
Rows # NULL 40.0 NULL
BIC NULL 27.041435998147332 NULL
(Intercept) NULL 0.3838162407664626 p

```

```
select * from admissions_factor_table;
```

```

masters_yes stats_beginner stats_novice programming_beginner programming_novice
gpa admitted

```

```

-----
-----
1      0      1      1      0 3.95      0.0
0      0      0      0      0 3.83      1.0
1      0      1      0      1 2.33      1.0
1      0      0      1      0 3.85      0.0
1      0      1      1      0 3.46      0.0
0      0      0      0      1 4.0       1.0
1      0      0      1      0 3.75      0.0
1      0      0      1      0 3.46      0.0
0      0      1      0      1 3.52      1.0
0      0      0      0      0 3.13      1.0
0      0      1      1      0 3.68      1.0
0      0      0      0      0 3.82      1.0
0      0      1      0      1 3.65      1.0
0      0      0      0      0 3.93      1.0
1      0      0      0      0 3.96      0.0
0      0      0      0      0 3.7       1.0
1      1      0      0      1 3.5       1.0
0      0      1      0      1 3.55      1.0
1      0      0      0      0 1.98      0.0
0      0      0      0      0 3.71      1.0
0      0      0      0      1 3.0       0.0
1      0      0      1      0 2.65      1.0
1      0      0      0      0 4.0       1.0
1      0      0      0      0 3.57      1.0
1      0      0      0      1 3.79      0.0
0      0      1      0      1 3.44      0.0
1      0      0      0      0 3.45      0.0
0      0      0      0      1 1.87      1.0
1      0      0      1      0 3.5       1.0
0      0      1      1      0 3.7       1.0

```

1	0	1	1	0	4.0	0.0
1	1	0	1	0	3.95	0.0
1	1	0	0	0	3.5	1.0
1	0	0	0	0	3.9	1.0
1	0	0	0	1	3.59	1.0
1	0	0	0	0	3.81	1.0
0	0	1	1	0	3.87	1.0
0	1	0	0	0	3.6	1.0
0	0	0	0	0	3.96	1.0
1	1	0	1	0		
3.76	0.0					

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLML1L2 Example: Factor Table as InputTable

### Input

The input table is `admissions_factor_table`, output by [GLML1L2 Example: Ridge Regression, Family \('BINOMIAL'\)](#). In `admissions_factor_table`, categorical predictors were converted to integers.

### SQL Call

Because the `admissions_factor_table` has only integers, this call does not specify `CategoricalColumns`.

```
DROP TABLE admissions_model_2;

CREATE MULTISET TABLE admissions_model_2 AS (
  SELECT * FROM GLML1L2 (
    ON admissions_factor_table AS InputTable
    USING
      TargetColumns ('[0:5]')
      ResponseColumn ('admitted')
      Family ('BINOMIAL')
      Alpha (0)
      RegularizationLambda (0.02)
  ) AS dt
) WITH DATA;
```

### Output

```
SELECT * FROM admissions_model_2;
```

attribute	estimate	information
-----	-----	-----

AIC	15.21927981934978	NULL
Iterations #	28.0	NULL
Features #	6.0	NULL
Alpha	0.0	NULL
Lambda	0.02	NULL
stats_novice	-0.026716553307241597	p
programming_novice	-0.0820786516340258	p
Regularization	NULL	Ridge
Family	NULL	Binomial
Converged	NULL	true
gpa	0.38346423433872745	p
Rows #	40.0	NULL
BIC	27.041435998147332	NULL
(Intercept)	0.3838162407664626	p
programming_beginner	-1.0259430213730834	p
stats_beginner	0.08063465501463249	p
masters_yes	-1.2652530272653697	p

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLML1L2 Example: LASSO, Family ('GAUSSIAN')

### Input

The input table is housing\_train, as in [GLM Example: Gaussian Distribution Analysis](#).

### SQL Call

Because the response variable has a Gaussian distribution, the call specifies Family ('GAUSSIAN'). Alpha (1) indicates L1 (LASSO) regularization.

```
DROP TABLE housing_model;
DROP TABLE housing_factor_table;

CREATE MULTISET TABLE housing_model AS (
  SELECT * FROM GLML1L2 (
    ON housing_train AS InputTable
    OUT TABLE FactorTable (housing_factor_table)
    USING
    TargetColumns
    ('lotsize','bedrooms','bathrms','stories','garagepl','driveway',
     'recroom','fullbase','gashw','airco','prefarea','homestyle')
    CategoricalColumns
    ('driveway','recroom','fullbase','gashw','airco','prefarea','homestyle')
```

```

    ResponseColumn ('price')
    Family ('GAUSSIAN')
    Alpha (1)
    RegularizationLambda (0.02)
  ) AS dt
) WITH DATA;

```

## Output

```
SELECT * FROM housing_model;
```

attribute	category	estimate	information
-----	-----	-----	-----
AIC	NULL	-8.992753699712395	NULL
Iterations #	NULL	53.0	NULL
stories	NULL	2445.6699824701327	p
bathrms	NULL	6772.387864268141	p
homestyle	classic	-54128.100924908955	p
Alpha	NULL	1.0	NULL
homestyle	eclectic	-37577.28812836616	p
prefarea	yes	6465.606895601109	p
Features #	NULL	13.0	NULL
airco	yes	6478.800674992518	p
Lambda	NULL	0.02	NULL
fullbase	yes	3588.313479183023	p
Family	NULL	NULL	Gaussian
recroom	yes	1208.5016271782713	p
Converged	NULL	NULL	true
bedrooms	NULL	782.1333264902945	p
Rows #	NULL	492.0	NULL
gashw	yes	5787.049927907821	p
driveway	yes	2822.6466305663917	p
Regularization	NULL	NULL	Lasso
RMSE	NULL	10385.734127243657	NULL
lotsize	NULL	2.080984013010899	p
BIC	NULL	49.78594833117991	NULL
(Intercept)	NULL	71103.48563681456	p
garagepl	NULL	1483.1186058015555	p

(housing\_factor\_table not shown here.)

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLML1L2Predict

The GLML1L2Predict function uses the model output by the [GLML1L2](#) function to perform generalized linear model prediction on new input data.

## GLML1L2Predict Syntax

### Version 1.7

```
SELECT * FROM GLML1L2Predict (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION
  USING
  [ OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Responses ('response' [,...]) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

## GLML1L2Predict Syntax Elements

### OutputProb

Specify whether to output the calculated probability for each observation.

Default: 'false'

### Responses

[Optional with OutputProb ('true'), disallowed otherwise] Specify the labels for which to output probabilities. A label (*response*) must be 0 or 1.

Default behavior: The function outputs only the probability of the class "1".

### Accumulate

Specify the names of input columns to copy to the output table.

## GLML1L2Predict Input

### Input Table Schema

#### Note:

It is important to normalize the input variables before calling this function. For details, see [Normalized Input](#).

Column	Data Type	Description
<i>feature_column</i>	Any numeric SQL data type or VARCHAR	Copied from InputTable used in GLML1L2 call that output Model table.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

## GLML1L2Predict Output

### Output Table Schema

The table is a set of predictions for each test point.

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.
prediction	DOUBLE PRECISION	Predicted value.
prob	DOUBLE PRECISION	[Column appears only if you specify OutputProb ('true') and omit Responses.] Probability that observation belongs to class "1". This value is $\text{sigmoid}(X\beta + b)$ , where $X$ is set of predictors and $\beta$ and $b$ are, respectively, vector of coefficients and intercept estimated by GLML1L2 function.  <b>Note:</b> If GLML1L2 call that created model specified Family ('GAUSSIAN'), all values in this column are NULL.
prob_0	DOUBLE PRECISION	[Only with Responses, one column appears for each <i>response</i> .] Probability that observation belongs to category 0. This value is $1 - \text{prob}$ , where <i>prob</i> is value in column prob.
prob_1	DOUBLE PRECISION	[Only with Responses, one column appears for each <i>response</i> .] Probability that observation belongs to category 1. This value is value in column prob.

## GLML1L2Predict Examples

### GLML1L2Predict Example: Ridge Regression, Family ('BINOMIAL')

#### Input

The input table is `admissions_test`, as in GLMPredict\_MLE [GLMPredict\\_MLE Example: Logistic Distribution Prediction](#).

The model table is `admissions_model`, output of [GLML1L2 Example: Ridge Regression, Family \('BINOMIAL'\)](#).

#### SQL Call

```
SELECT * FROM GLML1L2Predict (
  ON admissions_test PARTITION BY ANY
  ON admissions_model AS Model DIMENSION
```

```

USING
  OutputProb ('true')
  Accumulate ('id')
) AS dt ORDER BY id;

```

## Output

id	prediction	prob
50	0.0	0.42261164778201094
51	0.0	0.4049408918235976
52	1.0	0.6791731064692488
53	1.0	0.6128348254047428
54	1.0	0.6321200551583135
55	1.0	0.8635298008802573
56	1.0	0.8639684744611825
57	1.0	0.8589345247253525
58	1.0	0.8297786450140937
59	1.0	0.8421968614075399
60	1.0	0.8624268462528424
61	1.0	0.6575556307139581
62	1.0	0.8584692566063056
63	1.0	0.8644185196041072
64	1.0	0.640966603964
65	1.0	0.6488695133874245
66	1.0	0.693208669498573
67	0.0	0.35267304201513183
68	1.0	0.7347418494287773
69	1.0	0.8701555291596779

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## GLML1L2Predict Example: LASSO, Family ('GAUSSIAN')

### Input

- Input table: housing\_test, as in [GLMPredict\\_MLE Example: Gaussian Distribution Prediction](#)
- Model: housing\_model, output of [GLML1L2 Example: LASSO, Family \('GAUSSIAN'\)](#)

### SQL Call

```

SELECT * FROM GLML1L2Predict (
  ON housing_test PARTITION BY ANY
  ON housing_model AS Model DIMENSION

```

```

USING
  Accumulate ('sn')
) AS dt ORDER BY sn;

```

## Output

sn	prediction
13	37345.831973269815
16	43687.09059862308
25	40902.03654672491
38	72487.79993489318
53	79238.81946777017
104	111527.51245186775
111	39102.91738815808
117	66936.75897612597
132	41819.94760124324
140	41611.84919994215
142	44394.19887061475
157	66571.23189229079
161	64900.96267762861
162	107758.84817140205
176	73438.6189187027
177	62378.41894444322
195	37197.885003565505
198	47307.96151089168
224	67232.91218470069
234	35237.08241678744
237	46593.6202622758
239	43377.85857200969
249	37863.799887728994
251	45096.43470435488
254	74665.80555260641
255	60214.195570911885
260	43066.25992025624
274	67232.40606270777
294	38987.53125475488
301	55621.583411818116
306	67339.67883609686
317	65655.23197768882
329	123611.91388294056
339	126281.82379649683
340	58474.80959843659
353	67485.79160065402

```
355 68425.99104737997
364 77422.31599327197
367 128556.01523545264
377 127201.67378365475
401 84040.85339019273
403 79857.24053097825
408 76218.4796684783
411 78179.19706290774
440 80549.3290271716
441 64670.20979822652
443 61703.972586623415
459 42818.2957274824
463 49293.44779989387
469 61779.16835629488
472 63766.92478426945
527 119762.09345887038
530 116119.0226758564
540 73146.29611369228
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

# Linear regression

---

In statistics, **linear regression** is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called **multiple linear regression**.<sup>[1]</sup> This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.<sup>[2]</sup>

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.<sup>[3]</sup> Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications.<sup>[4]</sup> This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

- If the goal is prediction, forecasting, or error reduction, linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables. After developing such a model, if additional values of the explanatory variables are collected without an accompanying response value, the fitted model can be used to make a prediction of the response.
- If the goal is to explain variation in the response variable that can be attributed to variation in the explanatory variables, linear regression analysis can be applied to quantify the strength of the relationship between the response and the explanatory variables, and in particular to determine whether some explanatory variables may have no linear relationship with the response at all, or to identify which subsets of explanatory variables may contain redundant information about the response.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression ( $L^2$ -norm penalty) and lasso ( $L^1$ -norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms "least squares" and "linear model" are closely linked, they are not synonymous.

## Contents

---

### Introduction

### Assumptions

Interpretation**Extensions**Simple and multiple linear regressionGeneral linear modelsHeteroscedastic modelsGeneralized linear modelsHierarchical linear modelsErrors-in-variablesOthers**Estimation methods**Least-squares estimation and related techniquesMaximum-likelihood estimation and related techniquesOther estimation techniques**Applications**Trend lineEpidemiologyFinanceEconomicsEnvironmental scienceMachine learning**History****See also****References**CitationsSources**Further reading****External links**

## Introduction

---

Given a data set  $\{y_i, x_{i1}, \dots, x_{ip}\}_{i=1}^n$  of  $n$  statistical units, a linear regression model assumes that the relationship between the dependent variable  $y$  and the  $p$ -vector of regressors  $\mathbf{x}$  is linear. This relationship is modeled through a *disturbance term* or *error variable*  $\varepsilon$  — an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i = 1, \dots, n,$$

where <sup>T</sup> denotes the transpose, so that  $\mathbf{x}_i^T \boldsymbol{\beta}$  is the inner product between vectors  $\mathbf{x}_i$  and  $\boldsymbol{\beta}$ .

Often these  $n$  equations are stacked together and written in matrix notation as

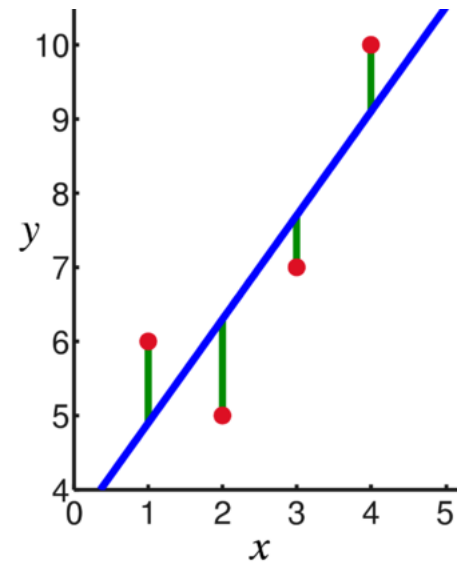
$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix},$$

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$



In linear regression, the observations (red) are assumed to be the result of random deviations (green) from an underlying relationship (blue) between a dependent variable ( $y$ ) and an independent variable ( $x$ ).

Some remarks on notation and terminology:

- $\mathbf{y}$  is a vector of observed values  $y_i$  ( $i = 1, \dots, n$ ) of the variable called the *regressand*, *endogenous variable*, *response variable*, *measured variable*, *criterion variable*, or *dependent variable*. This variable is also sometimes known as the *predicted variable*, but this should not be confused with *predicted values*, which are denoted  $\hat{y}$ . The decision as to which variable in a data set is modeled as the dependent variable and which are modeled as the independent variables may be based on a presumption that the value of one of the variables is caused by, or directly influenced by the other variables. Alternatively, there may be an operational reason to model one of the variables in terms of the others, in which case there need be no presumption of causality.
- $\mathbf{X}$  may be seen as a matrix of row-vectors  $\mathbf{x}_i$  or of  $n$ -dimensional column-vectors  $\mathbf{X}_j$ , which are known as *regressors*, *exogenous variables*, *explanatory variables*, *covariates*, *input variables*, *predictor variables*, or *independent variables* (not to be confused with the concept of *independent random variables*). The matrix  $\mathbf{X}$  is sometimes called the *design matrix*.
  - Usually a constant is included as one of the regressors. In particular,  $x_{i0} = 1$  for  $i = 1, \dots, n$ . The corresponding element of  $\boldsymbol{\beta}$  is called the *intercept*. Many statistical inference procedures for linear models require an intercept to be present, so it is often included even if theoretical considerations suggest that its value should be zero.
  - Sometimes one of the regressors can be a non-linear function of another regressor or of the data, as in *polynomial regression* and *segmented regression*. The model remains linear as long as it is linear in the parameter vector  $\boldsymbol{\beta}$ .
  - The values  $x_{ij}$  may be viewed as either observed values of random variables  $X_j$  or as fixed values chosen prior to observing the dependent variable. Both interpretations may be appropriate in different cases, and they generally lead to the same estimation procedures; however different approaches to asymptotic analysis are used in these two situations.
- $\boldsymbol{\beta}$  is a  $(p + 1)$ -dimensional *parameter vector*, where  $\beta_0$  is the intercept term (if one is included in the model—otherwise  $\boldsymbol{\beta}$  is  $p$ -dimensional). Its elements are known as *effects* or *regression coefficients*.

(although the latter term is sometimes reserved for the *estimated* effects). Statistical estimation and inference in linear regression focuses on  $\beta$ . The elements of this parameter vector are interpreted as the partial derivatives of the dependent variable with respect to the various independent variables.

- $\epsilon$  is a vector of values  $\epsilon_i$ . This part of the model is called the *error term*, *disturbance term*, or sometimes *noise* (in contrast with the "signal" provided by the rest of the model). This variable captures all other factors which influence the dependent variable  $y$  other than the regressors  $\mathbf{x}$ . The relationship between the error term and the regressors, for example their correlation, is a crucial consideration in formulating a linear regression model, as it will determine the appropriate estimation method.

Fitting a linear model to a given data set usually requires estimating the regression coefficients  $\beta$  such that the error term  $\epsilon = \mathbf{y} - \mathbf{X}\beta$  is minimized. For example, it is common to use the sum of squared errors  $\|\epsilon\|$  as the quality of the fit.

**Example.** Consider a situation where a small ball is being tossed up in the air and then we measure its heights of ascent  $h_i$  at various moments in time  $t_i$ . Physics tells us that, ignoring the drag, the relationship can be modeled as

$$h_i = \beta_1 t_i + \beta_2 t_i^2 + \epsilon_i,$$

where  $\beta_1$  determines the initial velocity of the ball,  $\beta_2$  is proportional to the standard gravity, and  $\epsilon_i$  is due to measurement errors. Linear regression can be used to estimate the values of  $\beta_1$  and  $\beta_2$  from the measured data. This model is non-linear in the time variable, but it is linear in the parameters  $\beta_1$  and  $\beta_2$ ; if we take regressors  $\mathbf{x}_i = (x_{i1}, x_{i2}) = (t_i, t_i^2)$ , the model takes on the standard form

$$h_i = \mathbf{x}_i^T \beta + \epsilon_i.$$

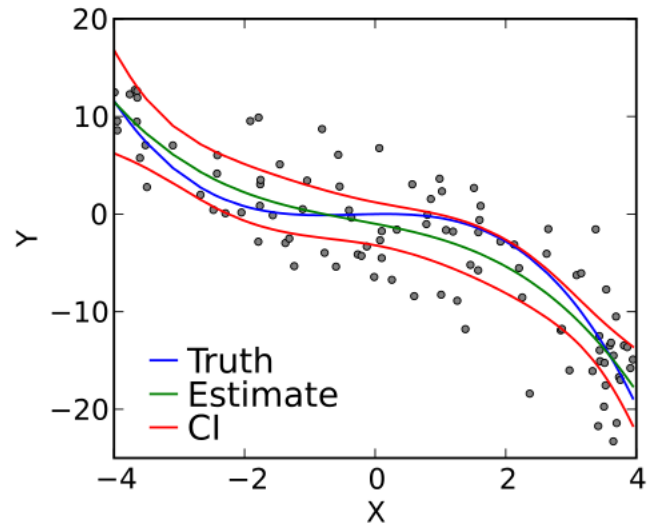
## Assumptions

Standard linear regression models with standard estimation techniques make a number of assumptions about the predictor variables, the response variables and their relationship. Numerous extensions have been developed that allow each of these assumptions to be relaxed (i.e. reduced to a weaker form), and in some cases eliminated entirely. Generally these extensions make the estimation procedure more complex and time-consuming, and may also require more data in order to produce an equally precise model.

The following are the major assumptions made by standard linear regression models with standard estimation techniques (e.g. ordinary least squares):

- **Weak exogeneity.** This essentially means that the predictor variables  $x$  can be treated as fixed values, rather than random variables. This means, for example, that the predictor variables are assumed to be error-free—that is, not contaminated with measurement errors. Although this assumption is not realistic in many settings, dropping it leads to significantly more difficult errors-in-variables models.

- **Linearity.** This means that the mean of the response variable is a linear combination of the parameters (regression coefficients) and the predictor variables. Note that this assumption is much less restrictive than it may at first seem. Because the predictor variables are treated as fixed values (see above), linearity is really only a restriction on the parameters. The predictor variables themselves can be arbitrarily transformed, and in fact multiple copies of the same underlying predictor variable can be added, each one transformed differently. This technique is used, for example, in polynomial regression, which uses linear regression to fit the response variable as an arbitrary polynomial function (up to a given rank) of a predictor variable. With this much flexibility, models such as polynomial regression often have "too much power", in that they tend to overfit the data. As a result, some kind of regularization must typically be used to prevent unreasonable solutions coming out of the estimation process. Common examples are ridge regression and lasso regression. Bayesian linear regression can also be used, which by its nature is more or less immune to the problem of overfitting. (In fact, ridge regression and lasso regression can both be viewed as special cases of Bayesian linear regression, with particular types of prior distributions placed on the regression coefficients.)
- **Constant variance** (a.k.a. **homoscedasticity**). This means that different values of the response variable have the same variance in their errors, regardless of the values of the predictor variables. In practice this assumption is invalid (i.e. the errors are heteroscedastic) if the response variable can vary over a wide scale. In order to check for heterogeneous error variance, or when a pattern of residuals violates model assumptions of homoscedasticity (error is equally variable around the 'best-fitting line' for all points of  $x$ ), it is prudent to look for a "fanning effect" between residual error and predicted values. This is to say there will be a systematic change in the absolute or squared residuals when plotted against the predictive variables. Errors will not be evenly distributed across the regression line. Heteroscedasticity will result in the averaging over of distinguishable variances around the points to get a single variance that is inaccurately representing all the variances of the line. In effect, residuals appear clustered and spread apart on their predicted plots for larger and smaller values for points along the linear regression line, and the mean squared error for the model will be wrong. Typically, for example, a response variable whose mean is large will have a greater variance than one whose mean is small. For example, a given person whose income is predicted to be \$100,000 may easily have an actual income of \$80,000 or \$120,000 (a standard deviation of around \$20,000), while another person with a predicted income of \$10,000 is unlikely to have the same \$20,000 standard deviation, which would imply their actual income would vary anywhere between -\$10,000 and \$30,000. (In fact, as this shows, in many cases—often the same cases where the assumption of normally distributed errors fails—the variance or standard deviation should be predicted to be proportional to the mean, rather than constant.) Simple linear regression estimation methods give less precise parameter estimates and misleading inferential quantities such as standard errors when substantial heteroscedasticity is present. However, various estimation techniques (e.g. weighted least squares and heteroscedasticity-consistent standard errors) can handle heteroscedasticity in a quite general way. Bayesian linear regression techniques can also be used when the variance is assumed to be a function of the mean. It is also possible in some cases to fix the problem by applying a transformation to the response variable (e.g. fit the logarithm of the response variable using a linear regression model, which implies that the response variable has a log-normal distribution rather than a normal distribution).



Example of a cubic polynomial regression, which is a type of linear regression.

- **Independence** of errors. This assumes that the errors of the response variables are uncorrelated with each other. (Actual statistical independence is a stronger condition than mere lack of correlation and is often not needed, although it can be exploited if it is known to hold.) Some methods (e.g. generalized least squares) are capable of handling correlated errors, although they typically require significantly more data unless some sort of regularization is used to bias the model towards assuming uncorrelated errors. Bayesian linear regression is a general way of handling this issue.
- **Lack of perfect multicollinearity** in the predictors. For standard least squares estimation methods, the design matrix  $X$  must have full column rank  $p$ ; otherwise, we have a condition known as perfect multicollinearity in the predictor variables. This can be triggered by having two or more perfectly correlated predictor variables (e.g. if the same predictor variable is mistakenly given twice, either without transforming one of the copies or by transforming one of the copies linearly). It can also happen if there is too little data available compared to the number of parameters to be estimated (e.g. fewer data points than regression coefficients). In the case of perfect multicollinearity, the parameter vector  $\beta$  will be non-identifiable—it has no unique solution. At most we will be able to identify some of the parameters, i.e. narrow down its value to some linear subspace of  $\mathbf{R}^p$ . See partial least squares regression. Methods for fitting linear models with multicollinearity have been developed;<sup>[5][6][7][8]</sup> some require additional assumptions such as "effect sparsity"—that a large fraction of the effects are exactly zero.  
Note that the more computationally expensive iterated algorithms for parameter estimation, such as those used in generalized linear models, do not suffer from this problem.

Beyond these assumptions, several other statistical properties of the data strongly influence the performance of different estimation methods:

- The statistical relationship between the error terms and the regressors plays an important role in determining whether an estimation procedure has desirable sampling properties such as being unbiased and consistent.
- The arrangement, or probability distribution of the predictor variables  $\mathbf{x}$  has a major influence on the precision of estimates of  $\beta$ . Sampling and design of experiments are highly developed subfields of statistics that provide guidance for collecting data in such a way to achieve a precise estimate of  $\beta$ .

## Interpretation

A fitted linear regression model can be used to identify the relationship between a single predictor variable  $x_j$  and the response variable  $y$  when all the other predictor variables in the model are "held fixed". Specifically, the interpretation of  $\beta_j$  is the expected change in  $y$  for a one-unit change in  $x_j$  when the other covariates are held fixed—that is, the expected value of the partial derivative of  $y$  with respect to  $x_j$ . This is sometimes called the *unique effect* of  $x_j$  on  $y$ . In contrast, the *marginal effect* of  $x_j$  on  $y$  can be assessed using a correlation coefficient or simple linear regression model relating only  $x_j$  to  $y$ ; this effect is the total derivative of  $y$  with respect to  $x_j$ .

Care must be taken when interpreting regression results, as some of the regressors may not allow for marginal changes (such as dummy variables, or the intercept term), while others cannot be held fixed (recall the example from the introduction: it would be impossible to "hold  $t_i$  fixed" and at the same time change the value of  $t_i^2$ ).

It is possible that the unique effect can be nearly zero even when the marginal effect is large. This may imply that some other covariate captures all the information in  $x_j$ , so that once that variable is in the model, there is no contribution of  $x_j$  to the variation in  $y$ . Conversely, the unique effect of  $x_j$  can be large while its marginal effect is nearly zero. This would happen if the other covariates explained a great deal

of the variation of  $y$ , but they mainly explain variation in a way that is complementary to what is captured by  $x_j$ . In this case, including the other variables in the model reduces the part of the variability of  $y$  that is unrelated to  $x_j$ , thereby strengthening the apparent relationship with  $x_j$ .

The meaning of the expression "held fixed" may depend on how the values of the predictor variables arise. If the experimenter directly sets the values of the predictor variables according to a study design, the comparisons of interest may literally correspond to comparisons among units whose predictor variables have been "held fixed" by the experimenter. Alternatively, the expression "held fixed" can refer to a selection that takes place in the context of data analysis. In this case, we "hold a variable fixed" by restricting our attention to the subsets of the data that happen to have a common value for the given predictor variable. This is the only interpretation of "held fixed" that can be used in an observational study.

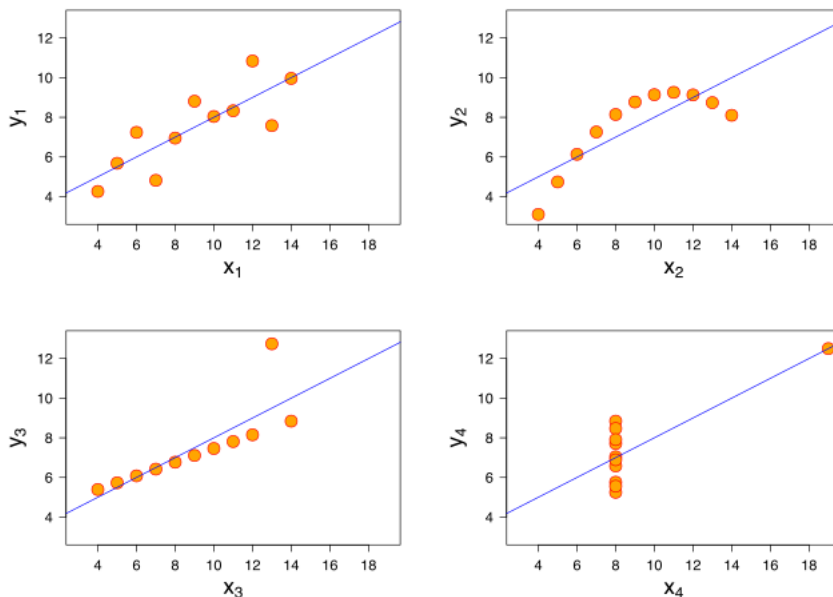
The notion of a "unique effect" is appealing when studying a complex system where multiple interrelated components influence the response variable. In some cases, it can literally be interpreted as the causal effect of an intervention that is linked to the value of a predictor variable. However, it has been argued that in many cases multiple regression analysis fails to clarify the relationships between the predictor variables and the response variable when the predictors are correlated with each other and are not assigned following a study design.<sup>[9]</sup> Commonality analysis may be helpful in disentangling the shared and unique impacts of correlated independent variables.<sup>[10]</sup>

## Extensions

Numerous extensions of linear regression have been developed, which allow some or all of the assumptions underlying the basic model to be relaxed.

### Simple and multiple linear regression

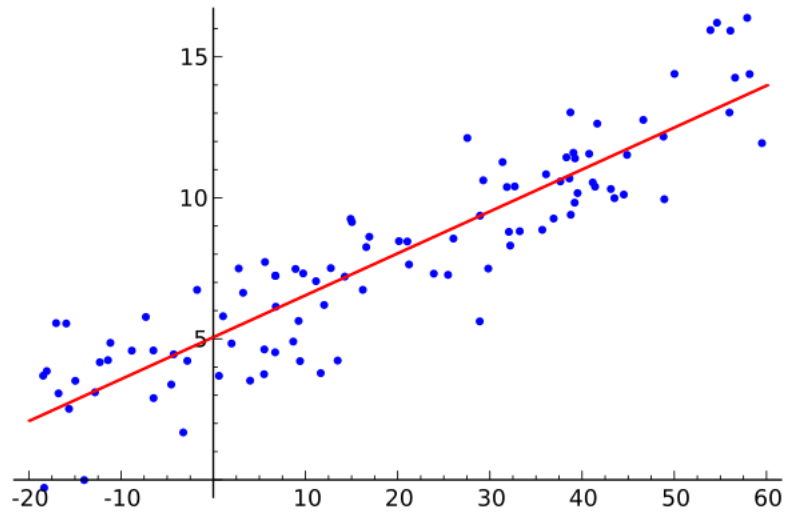
The very simplest case of a single scalar predictor variable  $x$  and a single scalar response variable  $y$  is known as *simple linear regression*. The extension to multiple and/or vector-valued predictor variables (denoted with a capital  $X$ ) is known as *multiple linear regression*, also known as *multivariable linear regression*. Nearly all real-world regression models involve multiple predictors, and basic descriptions of linear regression are often phrased in terms of the multiple regression model. Note, however, that in these cases the response variable  $y$  is still a scalar. Another term, *multivariate linear regression*, refers to cases where  $y$  is a vector, i.e., the same as *general linear regression*.



The data sets in the [Anscombe's quartet](#) are designed to have approximately the same linear regression line (as well as nearly identical means, standard deviations, and correlations) but are graphically very different. This illustrates the pitfalls of relying solely on a fitted model to understand the relationship between variables.

## General linear models

The general linear model considers the situation when the response variable is not a scalar (for each observation) but a vector,  $\mathbf{y}_i$ . Conditional linearity of  $E(\mathbf{y} | \mathbf{x}_i) = \mathbf{x}_i^T \mathbf{B}$  is still assumed, with a matrix  $\mathbf{B}$  replacing the vector  $\boldsymbol{\beta}$  of the classical linear regression model. Multivariate analogues of ordinary least squares (OLS) and generalized least squares (GLS) have been developed. "General linear models" are also called "multivariate linear models". These are not the same as multivariable linear models (also called "multiple linear models").



Example of simple linear regression, which has one independent variable

## Heteroscedastic models

Various models have been created that allow for heteroscedasticity, i.e. the errors for different response variables may have different variances. For example, weighted least squares is a method for estimating linear regression models when the response variables may have different error variances, possibly with correlated errors. (See also Weighted linear least squares, and Generalized least squares.) Heteroscedasticity-consistent standard errors is an improved method for use with uncorrelated but potentially heteroscedastic errors.

## Generalized linear models

Generalized linear models (GLMs) are a framework for modeling response variables that are bounded or discrete. This is used, for example:

- when modeling positive quantities (e.g. prices or populations) that vary over a large scale—which are better described using a skewed distribution such as the log-normal distribution or Poisson distribution (although GLMs are not used for log-normal data, instead the response variable is simply transformed using the logarithm function);
- when modeling categorical data, such as the choice of a given candidate in an election (which is better described using a Bernoulli distribution/binomial distribution for binary choices, or a categorical distribution/multinomial distribution for multi-way choices), where there are a fixed number of choices that cannot be meaningfully ordered;
- when modeling ordinal data, e.g. ratings on a scale from 0 to 5, where the different outcomes can be ordered but where the quantity itself may not have any absolute meaning (e.g. a rating of 4 may not be "twice as good" in any objective sense as a rating of 2, but simply indicates that it is better than 2 or 3 but not as good as 5).

Generalized linear models allow for an arbitrary *link function*,  $g$ , that relates the mean of the response variable(s) to the predictors:  $E(Y) = g^{-1}(XB)$ . The link function is often related to the distribution of the response, and in particular it typically has the effect of transforming between the  $(-\infty, \infty)$  range of the linear predictor and the range of the response variable.

Some common examples of GLMs are:

- Poisson regression for count data.
- Logistic regression and probit regression for binary data.
- Multinomial logistic regression and multinomial probit regression for categorical data.
- Ordered logit and ordered probit regression for ordinal data.

Single index models allow some degree of nonlinearity in the relationship between  $x$  and  $y$ , while preserving the central role of the linear predictor  $\beta'x$  as in the classical linear regression model. Under certain conditions, simply applying OLS to data from a single-index model will consistently estimate  $\beta$  up to a proportionality constant.<sup>[11]</sup>

## Hierarchical linear models

Hierarchical linear models (or *multilevel regression*) organizes the data into a hierarchy of regressions, for example where  $A$  is regressed on  $B$ , and  $B$  is regressed on  $C$ . It is often used where the variables of interest have a natural hierarchical structure such as in educational statistics, where students are nested in classrooms, classrooms are nested in schools, and schools are nested in some administrative grouping, such as a school district. The response variable might be a measure of student achievement such as a test score, and different covariates would be collected at the classroom, school, and school district levels.

## Errors-in-variables

Errors-in-variables models (or "measurement error models") extend the traditional linear regression model to allow the predictor variables  $X$  to be observed with error. This error causes standard estimators of  $\beta$  to become biased. Generally, the form of bias is an attenuation, meaning that the effects are biased toward zero.

## Others

- In Dempster–Shafer theory, or a linear belief function in particular, a linear regression model may be represented as a partially swept matrix, which can be combined with similar matrices representing observations and other assumed normal distributions and state equations. The combination of swept or unswept matrices provides an alternative method for estimating linear regression models.

## Estimation methods

---

A large number of procedures have been developed for parameter estimation and inference in linear regression. These methods differ in computational simplicity of algorithms, presence of a closed-form solution, robustness with respect to heavy-tailed distributions, and theoretical assumptions needed to validate desirable statistical properties such as consistency and asymptotic efficiency.

Some of the more common estimation techniques for linear regression are summarized below.

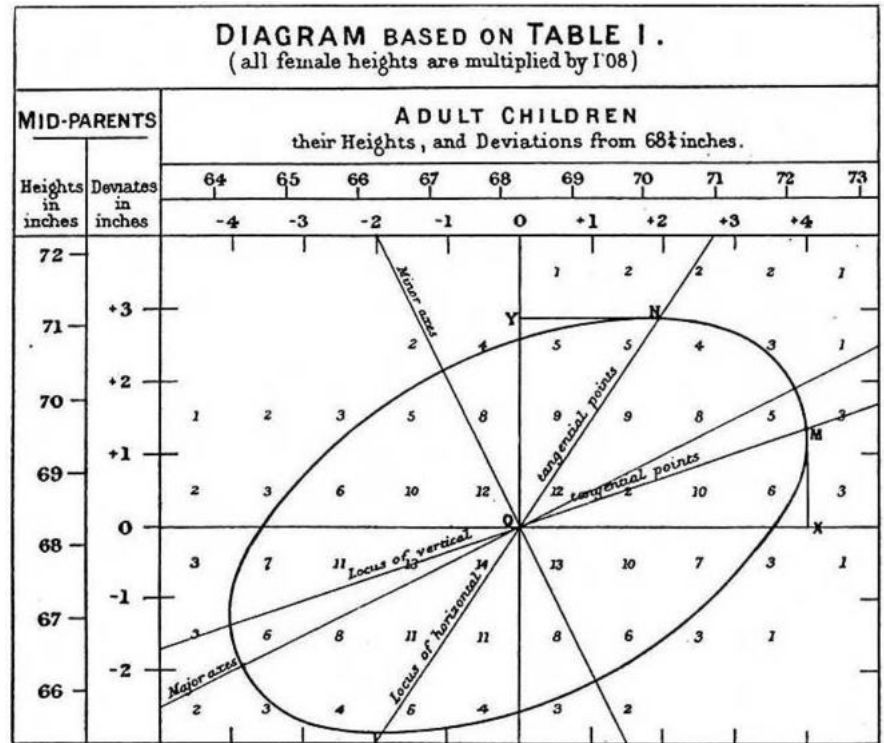
## Least-squares estimation and related techniques

Let's assume that the independent variable is  $\vec{x}_i = [x_1^i, x_2^i, \dots, x_m^i]$  and the model's parameters are  $\vec{\beta} = [\beta_0, \beta_1, \dots, \beta_m]$ , then the model's prediction would be  $y_i \approx \beta_0 + \sum_{j=1}^m \beta_j \times x_j^i$ . If  $\vec{x}_i$  is

extended to  $\vec{x}_i = [1, x_1^i, x_2^i, \dots, x_m^i]$  then  $y_i$  would become a dot product of the parameter and the independent variable, i.e.

$$y_i \approx \sum_{j=0}^m \beta_j \times x_j^i = \vec{\beta} \cdot \vec{x}_i. \text{ In the}$$

least-squares setting, the optimum parameter is defined as such that minimizes the sum of mean squared loss:



Francis Galton's 1875 illustration of the correlation between the heights of adults and their parents. The observation that adult children's heights tended to deviate less from the mean height than their parents suggested the concept of "regression toward the mean", giving regression its name. The "locus of horizontal tangential points" passing through the leftmost and rightmost points on the ellipse (which is a level curve of the bivariate normal distribution estimated from the data) is the OLS estimate of the regression of parents' heights on children's heights, while the "locus of vertical tangential points" is the OLS estimate of the regression of children's heights on parent's heights. The major axis of the ellipse is the TLS estimate.

$$\vec{\hat{\beta}} = \arg \min_{\vec{\beta}} L(D, \vec{\beta}) = \arg \min_{\vec{\beta}} \sum_{i=1}^n (\vec{\beta} \cdot \vec{x}_i - y_i)^2$$

Now putting the independent and dependent variables in matrices  $\mathbf{X}$  and  $\mathbf{Y}$  respectively, the loss function can be rewritten as:

$$\begin{aligned} L(D, \vec{\beta}) &= \|\mathbf{X}\vec{\beta} - \mathbf{Y}\|^2 \\ &= (\mathbf{X}\vec{\beta} - \mathbf{Y})^T (\mathbf{X}\vec{\beta} - \mathbf{Y}) \\ &= \mathbf{Y}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X} \vec{\beta} - \vec{\beta}^T \mathbf{X}^T \mathbf{Y} + \vec{\beta}^T \mathbf{X}^T \mathbf{X} \vec{\beta} \end{aligned}$$

As the loss is convex the optimum solution lies at gradient zero. The gradient of the loss function is (using Denominator layout convention):

$$\begin{aligned}\frac{\partial L(D, \vec{\beta})}{\partial \vec{\beta}} &= \frac{\partial (Y^T Y - Y^T X \vec{\beta} - \vec{\beta}^T X^T Y + \vec{\beta}^T X^T X \vec{\beta})}{\partial \vec{\beta}} \\ &= -2Y^T X + 2\vec{\beta}^T X^T X\end{aligned}$$

Setting the gradient to zero produces the optimum parameter:

$$\begin{aligned}-2Y^T X + 2\vec{\beta}^T X^T X &= 0 \\ \Rightarrow Y^T X &= \vec{\beta}^T X^T X \\ \Rightarrow X^T Y &= X^T X \vec{\beta} \\ \Rightarrow \hat{\vec{\beta}} &= (X^T X)^{-1} X^T Y\end{aligned}$$

**Note:** To prove that the  $\hat{\vec{\beta}}$  obtained is indeed the local minimum, one needs to differentiate once more to obtain the Hessian matrix and show that it is positive definite. This is provided by the Gauss–Markov theorem.

**Linear least squares** methods include mainly:

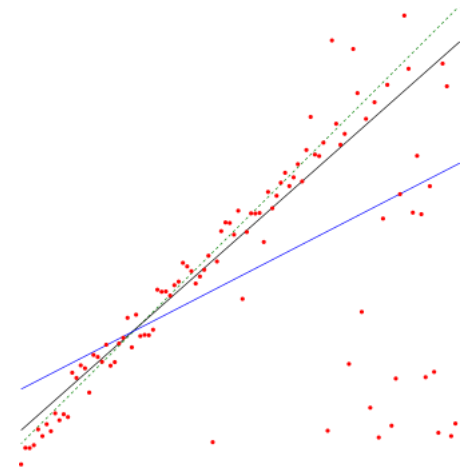
- Ordinary least squares
- Weighted least squares
- Generalized least squares

## Maximum-likelihood estimation and related techniques

- **Maximum likelihood estimation** can be performed when the distribution of the error terms is known to belong to a certain parametric family  $f_\theta$  of probability distributions.<sup>[12]</sup> When  $f_\theta$  is a normal distribution with zero mean and variance  $\theta$ , the resulting estimate is identical to the OLS estimate. GLS estimates are maximum likelihood estimates when  $\varepsilon$  follows a multivariate normal distribution with a known covariance matrix.
- **Ridge regression**<sup>[13][14][15]</sup> and other forms of penalized estimation, such as **Lasso regression**,<sup>[5]</sup> deliberately introduce bias into the estimation of  $\beta$  in order to reduce the variability of the estimate. The resulting estimates generally have lower mean squared error than the OLS estimates, particularly when multicollinearity is present or when overfitting is a problem. They are generally used when the goal is to predict the value of the response variable  $y$  for values of the predictors  $x$  that have not yet been observed. These methods are not as commonly used when the goal is inference, since it is difficult to account for the bias.
- **Least absolute deviation** (LAD) regression is a robust estimation technique in that it is less sensitive to the presence of outliers than OLS (but is less efficient than OLS when no outliers are present). It is equivalent to maximum likelihood estimation under a Laplace distribution model for  $\varepsilon$ .<sup>[16]</sup>
- **Adaptive estimation.** If we assume that error terms are independent of the regressors,  $\varepsilon_i \perp \mathbf{x}_i$ , then the optimal estimator is the 2-step MLE, where the first step is used to non-parametrically estimate the distribution of the error term.<sup>[17]</sup>

## Other estimation techniques

- **Bayesian linear regression** applies the framework of Bayesian statistics to linear regression. (See also [Bayesian multivariate linear regression](#).) In particular, the regression coefficients  $\beta$  are assumed to be random variables with a specified prior distribution. The prior distribution can bias the solutions for the regression coefficients, in a way similar to (but more general than) [ridge regression](#) or [lasso regression](#). In addition, the Bayesian estimation process produces not a single point estimate for the "best" values of the regression coefficients but an entire [posterior distribution](#), completely describing the uncertainty surrounding the quantity. This can be used to estimate the "best" coefficients using the mean, mode, median, any quantile (see [quantile regression](#)), or any other function of the posterior distribution.
- **Quantile regression** focuses on the conditional quantiles of  $y$  given  $X$  rather than the conditional mean of  $y$  given  $X$ . Linear quantile regression models a particular conditional quantile, for example the conditional median, as a linear function  $\beta^T x$  of the predictors.
- **Mixed models** are widely used to analyze linear regression relationships involving dependent data when the dependencies have a known structure. Common applications of mixed models include analysis of data involving repeated measurements, such as longitudinal data, or data obtained from cluster sampling. They are generally fit as parametric models, using maximum likelihood or Bayesian estimation. In the case where the errors are modeled as [normal](#) random variables, there is a close connection between mixed models and generalized least squares.<sup>[18]</sup> [Fixed effects estimation](#) is an alternative approach to analyzing this type of data.
- **Principal component regression** (PCR)<sup>[7][8]</sup> is used when the number of predictor variables is large, or when strong correlations exist among the predictor variables. This two-stage procedure first reduces the predictor variables using [principal component analysis](#) then uses the reduced variables in an OLS regression fit. While it often works well in practice, there is no general theoretical reason that the most informative linear function of the predictor variables should lie among the dominant principal components of the multivariate distribution of the predictor variables. The [partial least squares regression](#) is the extension of the PCR method which does not suffer from the mentioned deficiency.
- **Least-angle regression**<sup>[6]</sup> is an estimation procedure for linear regression models that was developed to handle high-dimensional covariate vectors, potentially with more covariates than observations.
- The **Theil–Sen estimator** is a simple robust estimation technique that chooses the slope of the fit line to be the median of the slopes of the lines through pairs of sample points. It has similar statistical efficiency properties to simple linear regression but is much less sensitive to [outliers](#).<sup>[19]</sup>
- Other robust estimation techniques, including the  **$\alpha$ -trimmed mean** approach, and **L-, M-, S-, and R-estimators** have been introduced.



Comparison of the [Theil–Sen estimator](#) (black) and [simple linear regression](#) (blue) for a set of points with outliers.

## Applications

Linear regression is widely used in biological, behavioral and social sciences to describe possible relationships between variables. It ranks as one of the most important tools used in these disciplines.

### Trend line

A **trend line** represents a trend, the long-term movement in time series data after other components have been accounted for. It tells whether a particular data set (say GDP, oil prices or stock prices) have increased or decreased over the period of time. A trend line could simply be drawn by eye through a set of data points, but more properly their position and slope is calculated using statistical techniques like linear regression. Trend lines typically are straight lines, although some variations use higher degree polynomials depending on the degree of curvature desired in the line.

Trend lines are sometimes used in business analytics to show changes in data over time. This has the advantage of being simple. Trend lines are often used to argue that a particular action or event (such as training, or an advertising campaign) caused observed changes at a point in time. This is a simple technique, and does not require a control group, experimental design, or a sophisticated analysis technique. However, it suffers from a lack of scientific validity in cases where other potential changes can affect the data.

## Epidemiology

Early evidence relating tobacco smoking to mortality and morbidity came from observational studies employing regression analysis. In order to reduce spurious correlations when analyzing observational data, researchers usually include several variables in their regression models in addition to the variable of primary interest. For example, in a regression model in which cigarette smoking is the independent variable of primary interest and the dependent variable is lifespan measured in years, researchers might include education and income as additional independent variables, to ensure that any observed effect of smoking on lifespan is not due to those other socio-economic factors. However, it is never possible to include all possible confounding variables in an empirical analysis. For example, a hypothetical gene might increase mortality and also cause people to smoke more. For this reason, randomized controlled trials are often able to generate more compelling evidence of causal relationships than can be obtained using regression analyses of observational data. When controlled experiments are not feasible, variants of regression analysis such as instrumental variables regression may be used to attempt to estimate causal relationships from observational data.

## Finance

The capital asset pricing model uses linear regression as well as the concept of beta for analyzing and quantifying the systematic risk of an investment. This comes directly from the beta coefficient of the linear regression model that relates the return on the investment to the return on all risky assets.

## Economics

Linear regression is the predominant empirical tool in economics. For example, it is used to predict consumption spending,<sup>[20]</sup> fixed investment spending, inventory investment, purchases of a country's exports,<sup>[21]</sup> spending on imports,<sup>[21]</sup> the demand to hold liquid assets,<sup>[22]</sup> labor demand,<sup>[23]</sup> and labor supply.<sup>[23]</sup>

## Environmental science

Linear regression finds application in a wide range of environmental science applications. In Canada, the Environmental Effects Monitoring Program uses statistical analyses on fish and benthic surveys to measure the effects of pulp mill or metal mine effluent on the aquatic ecosystem.<sup>[24]</sup>

## Machine learning

Linear regression plays an important role in the field of artificial intelligence such as machine learning. The linear regression algorithm is one of the fundamental supervised machine-learning algorithms due to its relative simplicity and well-known properties.<sup>[25]</sup>

## History

---

Least squares linear regression, as a means of finding a good rough linear fit to a set of points was performed by Legendre (1805) and Gauss (1809) for the prediction of planetary movement. Quetelet was responsible for making the procedure well-known and for using it extensively in the social sciences.<sup>[26]</sup>

## See also

---

- Analysis of variance
- Blinder–Oaxaca decomposition
- Censored regression model
- Cross-sectional regression
- Curve fitting
- Empirical Bayes methods
- Errors and residuals
- Lack-of-fit sum of squares
- Line fitting
- Linear classifier
- Linear equation
- Logistic regression
- M-estimator
- Multivariate adaptive regression splines
- Nonlinear regression
- Nonparametric regression
- Normal equations
- Projection pursuit regression
- Segmented linear regression
- Stepwise regression
- Structural break
- Support vector machine
- Truncated regression model

## References

---

### Citations

1. David A. Freedman (2009). *Statistical Models: Theory and Practice*. Cambridge University Press. p. 26. "A simple regression equation has on the right hand side an intercept and an explanatory variable with a slope coefficient. A multiple regression e right hand side, each with its own slope coefficient"
2. Rencher, Alvin C.; Christensen, William F. (2012), "Chapter 10, Multivariate regression – Section 10.1, Introduction", *Methods of Multivariate Analysis* (<https://books.google.com/books?id=0g-PAuKub3QC&pg=PA19>), Wiley Series in Probability and Statistics, **709** (3rd ed.), John Wiley & Sons, p. 19, ISBN 9781118391679.
3. Hilary L. Seal (1967). "The historical development of the Gauss linear model". *Biometrika*. **54** (1/2): 1–24. doi:10.1093/biomet/54.1-2.1 (<https://doi.org/10.1093%2Fbiomet%2F54.1-2.1>). JSTOR 2333849 (<https://www.jstor.org/stable/2333849>).

4. Yan, Xin (2009), *Linear Regression Analysis: Theory and Computing* (<https://books.google.com/books?id=MjNv6rGv8NIC&pg=PA1>), World Scientific, pp. 1–2, ISBN 9789812834119, "Regression analysis ... is probably one of the oldest topics in mathematical statistics dating back to about two hundred years ago. The earliest form of the linear regression was the least squares method, which was published by Legendre in 1805, and by Gauss in 1809 ... Legendre and Gauss both applied the method to the problem of determining, from astronomical observations, the orbits of bodies about the sun."
5. Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". *Journal of the Royal Statistical Society, Series B*. **58** (1): 267–288. JSTOR 2346178 (<https://www.jstor.org/stable/2346178>).
6. Efron, Bradley; Hastie, Trevor; Johnstone, Iain; Tibshirani, Robert (2004). "Least Angle Regression". *The Annals of Statistics*. **32** (2): 407–451. arXiv:math/0406456 (<https://arxiv.org/abs/math/0406456>). doi:10.1214/009053604000000067 (<https://doi.org/10.1214%2F009053604000000067>). JSTOR 3448465 (<https://www.jstor.org/stable/3448465>).
7. Hawkins, Douglas M. (1973). "On the Investigation of Alternative Regressions by Principal Component Analysis". *Journal of the Royal Statistical Society, Series C*. **22** (3): 275–286. JSTOR 2346776 (<https://www.jstor.org/stable/2346776>).
8. Jolliffe, Ian T. (1982). "A Note on the Use of Principal Components in Regression". *Journal of the Royal Statistical Society, Series C*. **31** (3): 300–303. JSTOR 2348005 (<https://www.jstor.org/stable/2348005>).
9. Berk, Richard A. (2007). "Regression Analysis: A Constructive Critique". *Criminal Justice Review*. **32** (3): 301–302. doi:10.1177/0734016807304871 (<https://doi.org/10.1177%2F0734016807304871>).
10. Warne, Russell T. (2011). "Beyond multiple regression: Using commonality analysis to better understand R2 results". *Gifted Child Quarterly*. **55** (4): 313–318. doi:10.1177/0016986211422217 (<https://doi.org/10.1177%2F0016986211422217>).
11. Brillinger, David R. (1977). "The Identification of a Particular Nonlinear Time Series System". *Biometrika*. **64** (3): 509–515. doi:10.1093/biomet/64.3.509 (<https://doi.org/10.1093%2Fbiomet%2F64.3.509>). JSTOR 2345326 (<https://www.jstor.org/stable/2345326>).
12. Lange, Kenneth L.; Little, Roderick J. A.; Taylor, Jeremy M. G. (1989). "Robust Statistical Modeling Using the t Distribution" (<https://cloudfront.escholarship.org/dist/prd/content/qt27s1d3h7/qt27s1d3h7.pdf>) (PDF). *Journal of the American Statistical Association*. **84** (408): 881–896. doi:10.2307/2290063 (<https://doi.org/10.2307%2F2290063>). JSTOR 2290063 (<https://www.jstor.org/stable/2290063>).
13. Swindel, Benée F. (1981). "Geometry of Ridge Regression Illustrated". *The American Statistician*. **35** (1): 12–15. doi:10.2307/2683577 (<https://doi.org/10.2307%2F2683577>). JSTOR 2683577 (<https://www.jstor.org/stable/2683577>).
14. Draper, Norman R.; van Nostrand, R. Craig (1979). "Ridge Regression and James-Stein Estimation: Review and Comments". *Technometrics*. **21** (4): 451–466. doi:10.2307/1268284 (<https://doi.org/10.2307%2F1268284>). JSTOR 1268284 (<https://www.jstor.org/stable/1268284>).
15. Hoerl, Arthur E.; Kennard, Robert W.; Hoerl, Roger W. (1985). "Practical Use of Ridge Regression: A Challenge Met". *Journal of the Royal Statistical Society, Series C*. **34** (2): 114–120. JSTOR 2347363 (<https://www.jstor.org/stable/2347363>).
16. Narula, Subhash C.; Wellington, John F. (1982). "The Minimum Sum of Absolute Errors Regression: A State of the Art Survey". *International Statistical Review*. **50** (3): 317–326. doi:10.2307/1402501 (<https://doi.org/10.2307%2F1402501>). JSTOR 1402501 (<https://www.jstor.org/stable/1402501>).
17. Stone, C. J. (1975). "Adaptive maximum likelihood estimators of a location parameter" (<https://doi.org/10.1214/aos/1176343056>). *The Annals of Statistics*. **3** (2): 267–284. doi:10.1214/aos/1176343056 (<https://doi.org/10.1214%2Faos%2F1176343056>). JSTOR 2958945 (<https://www.jstor.org/stable/2958945>).
18. Goldstein, H. (1986). "Multilevel Mixed Linear Model Analysis Using Iterative Generalized Least Squares". *Biometrika*. **73** (1): 43–56. doi:10.1093/biomet/73.1.43 (<https://doi.org/10.1093%2Fbiomet%2F73.1.43>). JSTOR 2336270 (<https://www.jstor.org/stable/2336270>).

19. Theil, H. (1950). "A rank-invariant method of linear and polynomial regression analysis. I, II, III". *Nederl. Akad. Wetensch., Proc.* **53**: 386–392, 521–525, 1397–1412. MR 0036489 (<https://www.ams.org/mathscinet-getitem?mr=0036489>).; Sen, Pranab Kumar (1968). "Estimates of the regression coefficient based on Kendall's tau". *Journal of the American Statistical Association*. **63** (324): 1379–1389. doi:10.2307/2285891 (<https://doi.org/10.2307%2F2285891>). JSTOR 2285891 (<https://www.jstor.org/stable/2285891>). MR 0258201 (<https://www.ams.org/mathscinet-getitem?mr=0258201>)..
20. Deaton, Angus (1992). *Understanding Consumption*. Oxford University Press. ISBN 978-0-19-828824-4.
21. Krugman, Paul R.; Obstfeld, M.; Melitz, Marc J. (2012). *International Economics: Theory and Policy* (9th global ed.). Harlow: Pearson. ISBN 9780273754091.
22. Laidler, David E. W. (1993). *The Demand for Money: Theories, Evidence, and Problems* (4th ed.). New York: Harper Collins. ISBN 978-0065010985.
23. Ehrenberg; Smith (2008). *Modern Labor Economics* (10th international ed.). London: Addison-Wesley. ISBN 9780321538963.
24. EEMP webpage (<http://www.ec.gc.ca/eseee-eem/default.asp?lang=En&n=453D78FC-1>) Archived (<https://web.archive.org/web/20110611114740/http://www.ec.gc.ca/eseee-eem/default.asp?lang=En&n=453D78FC-1>) 2011-06-11 at the Wayback Machine
25. "Linear Regression (Machine Learning)" (<https://people.cs.pitt.edu/~milos/courses/cs2750-Spring03/lectures/class6.pdf>) (PDF). *University of Pittsburgh*.
26. Stigler, Stephen M. (1986). *The History of Statistics: The Measurement of Uncertainty before 1900* (<https://archive.org/details/historyofstatist00stig>). Cambridge: Harvard. ISBN 0-674-40340-1.

## Sources

- Cohen, J., Cohen P., West, S.G., & Aiken, L.S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences* ([https://books.google.com/books?id=98p4AgAAQBAJ&printsec=frontcover&dq=%22Applied+multiple+regression/correlation+analysis+for+the+behavioral+science%22&hl=en&sa=X&ved=0ahUKEwjRwv\\_Z79fiAhUB\\_1QKHdBQCgoQ6AEIKjAA#v=onepage&q=%22Applied%20multiple%20regression%2Fcorrelation%20analysis%20for%20the%20behavioral%20sciences%22&f=false](https://books.google.com/books?id=98p4AgAAQBAJ&printsec=frontcover&dq=%22Applied+multiple+regression/correlation+analysis+for+the+behavioral+science%22&hl=en&sa=X&ved=0ahUKEwjRwv_Z79fiAhUB_1QKHdBQCgoQ6AEIKjAA#v=onepage&q=%22Applied%20multiple%20regression%2Fcorrelation%20analysis%20for%20the%20behavioral%20sciences%22&f=false)). (2nd ed.) Hillsdale, NJ: Lawrence Erlbaum Associates
- Charles Darwin. *The Variation of Animals and Plants under Domestication*. (1868) (Chapter XIII describes what was known about reversion in Galton's time. Darwin uses the term "reversion".)
- Draper, N.R.; Smith, H. (1998). *Applied Regression Analysis* (3rd ed.). John Wiley. ISBN 978-0-471-17082-2.
- Francis Galton. "Regression Towards Mediocrity in Hereditary Stature," *Journal of the Anthropological Institute*, 15:246-263 (1886). (Facsimile at: [1] (<http://www.mugu.com/galton/essays/1880-1889/galton-1886-jaigi-regression-stature.pdf>))
- Robert S. Pindyck and Daniel L. Rubinfeld (1998, 4h ed.). *Econometric Models and Economic Forecasts*, ch. 1 (Intro, incl. appendices on  $\Sigma$  operators & derivation of parameter est.) & Appendix 4.3 (mult. regression in matrix form).

## Further reading

- Pedhazur, Elazar J (1982). *Multiple regression in behavioral research: Explanation and prediction* (2nd ed.). New York: Holt, Rinehart and Winston. ISBN 978-0-03-041760-3.
- Mathieu Rouaud, 2013: Probability, Statistics and Estimation (<http://www.incertitudes.fr/book.pdf>) Chapter 2: Linear Regression, Linear Regression with Error Bars and Nonlinear Regression.
- National Physical Laboratory (1961). "Chapter 1: Linear Equations and Matrices: Direct Methods". *Modern Computing Methods*. Notes on Applied Science. **16** (2nd ed.). Her Majesty's Stationery

Office.

## External links

---

- [Least-Squares Regression \(https://phet.colorado.edu/en/simulation/least-squares-regression\)](https://phet.colorado.edu/en/simulation/least-squares-regression), PhET Interactive simulations, University of Colorado at Boulder
  - [DIY Linear Fit \(http://www.geocities.ws/diylf/DIYLF.html\)](http://www.geocities.ws/diylf/DIYLF.html)
- 

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Linear\\_regression&oldid=977730584](https://en.wikipedia.org/w/index.php?title=Linear_regression&oldid=977730584)"

---

**This page was last edited on 10 September 2020, at 16:37 (UTC).**

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Source: <https://www.thebalancesmb.com/what-is-simple-linear-regression-2296697>

# What A Simple Linear Regression Model Is and How It Works

A basic statistical method of finding relationships between variables

BY

GIGI DEVAULT

Updated January 09, 2020

Linear regression models are used to show or predict the relationship between two [variables or factors](#). The factor that is being predicted (the factor that the equation *solves for*) is called the dependent variable. The factors that are used to predict the value of the dependent variable are called the independent variables.

In linear regression, each [observation](#) consists of two values. One value is for the dependent variable and one value is for the independent variable. In [this simple model](#), a straight line approximates the relationship between the dependent variable and the independent variable.<sup>1</sup>

When two or more independent variables are used in regression analysis, the model is no longer a simple linear one. This is known as multiple regression.<sup>2</sup>

## Formula For a Simple Linear Regression Model

The two factors that are involved in simple linear regression analysis are designated  $x$  and  $y$ . The equation that describes how  $y$  is related to  $x$  is known as the **regression model**.

The simple linear regression model is represented by:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

The linear regression model contains an error term that is represented by  $\varepsilon$ . The error term is used to account for the variability in  $y$  that cannot be explained by the [linear relationship](#) between  $x$  and  $y$ . If  $\varepsilon$  were not present, that would mean that knowing  $x$  would provide enough information to determine the value of  $y$ .

There also parameters that represent the population being studied. These [parameters of the model](#) are represented by  $\beta_0$  and  $\beta_1$ .

The simple linear regression equation is graphed as a straight line, where:

1.  $\beta_0$  is the y-intercept of the regression line.
2.  $\beta_1$  is the slope.
3.  $E(y)$  is the mean or expected value of  $y$  for a given value of  $x$ .

A regression line can show a positive linear relationship, a negative linear relationship, or no relationship<sup>3</sup>.

1. **No relationship:** The graphed line in a simple linear regression is flat (not sloped). There is no relationship between the two variables.
2. **Positive relationship:** The regression line slopes upward with the lower end of the line at the y-intercept (axis) of the graph and the upper end of the line extending upward into the graph field, away from the x-intercept (axis). There is a positive linear relationship between the two variables: as the value of one increases, the value of the other also increases.
3. **Negative relationship:** The regression line slopes downward with the upper end of the line at the y-intercept (axis) of the graph and the lower end of the line extending downward into the graph field, toward the x-intercept (axis). There is a negative linear relationship between the two variables: as the value of one increases, the value of the other decreases.<sup>4</sup>

## The Estimated Linear Regression Equation

If the [parameters of the population](#) were known, the simple linear regression equation (shown below) could be used to compute the mean value of  $y$  for a known value of  $x$ .

$$E(y) = \beta_0 + \beta_1 x + \varepsilon$$

In practice, however, parameter values generally are not known so they must be estimated by using [data from a sample](#) of the population. The population parameters are estimated by using [sample statistics](#). The [sample statistics](#) are represented by  $\beta_0$  and  $\beta_1$ . When the sample statistics are substituted for the population parameters, the estimated regression equation is formed.<sup>3</sup>

The estimated regression equation is:

$$(\hat{y}) = \beta_0 + \beta_1 x + \varepsilon$$

*Note:  $(\hat{y})$  is pronounced y hat.*

The graph of the estimated simple regression equation is called the estimated regression line.

1.  $\beta_0$  is the y-intercept of the regression line.
2.  $\beta_1$  is the slope.
3.  $(\hat{y})$  is the estimated value of  $y$  for a given value of  $x$ .

## Limits of Simple Linear Regression

Even the best data does not tell a complete story.

Regression analysis is commonly used in research to establish that a correlation exists between variables. But [correlation is not the same as causation](#): a relationship between two variables does not mean one causes the other to happen. Even a line in a simple linear regression that fits the data points well may not guarantee a cause-and-effect relationship.

Using a linear regression model will allow you to discover whether a relationship between variables exists at all. To understand exactly what that relationship is, and whether one variable causes another, you will need additional research and statistical analysis.<sup>1</sup>

### Article Sources

1. Anderson, D. R., Sweeney, D. J., and Williams, T. A. "Essentials of Statistics for Business and Economics (3rd edition)." Accessed January 8, 2020.
2. North Carolina State University. Using Cigarette Data for An Introduction to Multiple Regression. *Journal of Statistics Education*, 2(1). Accessed January 8, 2020.
3. Massachusetts Institute of Technology: MIT OpenCourseWare. "Statistics for Applications: Simple Linear Regression." Accessed January 8, 2020.
4. Mendenhall, W., and Sincich, T. (1992). "Statistics for Engineering and the Sciences (5th edition)." Accessed January 8, 2020.

# 4.0 Data Analytics Methods & Algorithms

ACCESSING THIS CUSTOM DOCUMENT BEAR THE ENTIRE RISK OF ANY RELIANCE ON THIS CUSTOM DOCUMENT, INCLUDING AS TO QUALITY, ACCURACY, AND RESULTS.

## POSTagger (ML Engine)

The POSTagger function creates part-of-speech (POS) tags for the words in the input text. POS tagging is the first step in the syntactic analysis of a language, and an important preprocessing step in many natural language-processing applications.

The POSTagger function was developed on the Penn Treebank Project and Chinese Penn Treebank Project data set. Its POS tags comply with the tags defined by the two projects.

For the parts of speech used, see the following:

Text Language	Parts of Speech
English	<a href="https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html">https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html</a>
Chinese	<a href="https://www.sketchengine.co.uk/chinese-penn-treebank-part-of-speech-tagset/">https://www.sketchengine.co.uk/chinese-penn-treebank-part-of-speech-tagset/</a>

POSTagger uses files that are preinstalled on ML Engine. For details, see Preinstalled Files That Functions Use.

## POSTagger Syntax

### Version 2.8

```
SELECT * FROM POSTagger (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')]
  [ InputLanguage ({ 'en' | 'zh_Cn' }) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related information

Related information

Column Specification Syntax Elements

## POSTagger Syntax Elements

TextColumn

Specify the name of the input column that contains the text to tag.

InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese

Accumulate

[Optional] Specify the names of the input table columns to copy to the output table.

If you intend to use the POSTagger output table as input to the function TextChunker (ML Engine), then this syntax element must specify the input table columns that comprise the partition key.

## POSTagger Input

Table	Description								
Input table	Contains text to tag.								
Model table	Determined by InputLanguage syntax element: <table> <tr> <th>InputLanguage</th><th>Model File</th></tr> <tr> <td>English</td><td>pos_model_2.0_en_141008.bin</td></tr> <tr> <td>Simplified</td><td>pos_model_2.0_zh_cn_141008.bin</td></tr> <tr> <td>Chinese</td><td></td></tr> </table> <p>These model files are preinstalled on ML Engine.</p>	InputLanguage	Model File	English	pos_model_2.0_en_141008.bin	Simplified	pos_model_2.0_zh_cn_141008.bin	Chinese	
InputLanguage	Model File								
English	pos_model_2.0_en_141008.bin								
Simplified	pos_model_2.0_zh_cn_141008.bin								
Chinese									

## Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
accumulate_column	Any	Column to copy to output table.
text_column	VARCHAR	Text to tag. Each row of this column must contain a well-formatted sentence. To convert English text to formatted sentences, use SentenceExtractor (ML Engine) function.

## POSTagger Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as in input table	[Column appears once for each specified accumulate_column.] Column copied from input table.
word_sn	INTEGER	Word serial number (position of word in input text).
word	VARCHAR	Word extracted from input text.
pos_tag	VARCHAR	POS tag of word.

## POSTagger Example

### Input

- Input table: Output table of SentenceExtractor Example

### SQL Call

```
SELECT * FROM POSTagger (
  ON SentenceExtractor (
    ON paragraphs_input
    USING
    TextColumn ('paratext')
    Accumulate ('paraid')
  )
  USING
  TextColumn ('sentence')
  Accumulate ('sentence', 'sentence_sn')
) AS dt ORDER BY sentence_sn, word_sn;
```

### Output

sentence

-----

in statistics, simple linear regression is the least squares estimator of a linear regression model v  
 logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
 association rule learning is a method for discovering interesting relations between variables in large  
 decision tree learning uses a decision tree as a predictive model which maps observations about an in  
 cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
 cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
 logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
 decision tree learning uses a decision tree as a predictive model which maps observations about an in  
 association rule learning is a method for discovering interesting relations between variables in large  
 in statistics, simple linear regression is the least squares estimator of a linear regression model v  
 logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
 association rule learning is a method for discovering interesting relations between variables in large  
 in statistics, simple linear regression is the least squares estimator of a linear regression model v  
 decision tree learning uses a decision tree as a predictive model which maps observations about an in  
 cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
 decision tree learning uses a decision tree as a predictive model which maps observations about an in  
 in statistics, simple linear regression is the least squares estimator of a linear regression model v  
 cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
decision tree learning uses a decision tree as a predictive model which maps observations about an instance  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model  
in statistics, simple linear regression is the least squares estimator of a linear regression model

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done in statistics, simple linear regression is the least squares estimator of a linear regression model v

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

decision tree learning uses a decision tree as a predictive model which maps observations about an in

association rule learning is a method for discovering interesting relations between variables in larg

in statistics, simple linear regression is the least squares estimator of a linear regression model v

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

association rule learning is a method for discovering interesting relations between variables in larg

decision tree learning uses a decision tree as a predictive model which maps observations about an in

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

decision tree learning uses a decision tree as a predictive model which maps observations about an in

association rule learning is a method for discovering interesting relations between variables in larg

in statistics, simple linear regression is the least squares estimator of a linear regression model v

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

in statistics, simple linear regression is the least squares estimator of a linear regression model v

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

association rule learning is a method for discovering interesting relations between variables in larg

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

decision tree learning uses a decision tree as a predictive model which maps observations about an in

in statistics, simple linear regression is the least squares estimator of a linear regression model v

association rule learning is a method for discovering interesting relations between variables in larg

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

decision tree learning uses a decision tree as a predictive model which maps observations about an in

association rule learning is a method for discovering interesting relations between variables in larg

in statistics, simple linear regression is the least squares estimator of a linear regression model v

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

in statistics, simple linear regression is the least squares estimator of a linear regression model v

decision tree learning uses a decision tree as a predictive model which maps observations about an in

association rule learning is a method for discovering interesting relations between variables in larg

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

decision tree learning uses a decision tree as a predictive model which maps observations about an in

in statistics, simple linear regression is the least squares estimator of a linear regression model v

association rule learning is a method for discovering interesting relations between variables in larg

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in

logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

in statistics, simple linear regression is the least squares estimator of a linear regression model v

decision tree learning uses a decision tree as a predictive model which maps observations about an in

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
association rule learning is a method for discovering interesting relations between variables in large  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large

[illegible]

in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
association rule learning is a method for discovering interesting relations between variables in large  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an i  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
in statistics, simple linear regression is the least squares estimator of a linear regression model w  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
logistic regression was developed by statistician david cox in 1958[2][3](although much work was done  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in  
cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in  
association rule learning is a method for discovering interesting relations between variables in large  
decision tree learning uses a decision tree as a predictive model which maps observations about an in

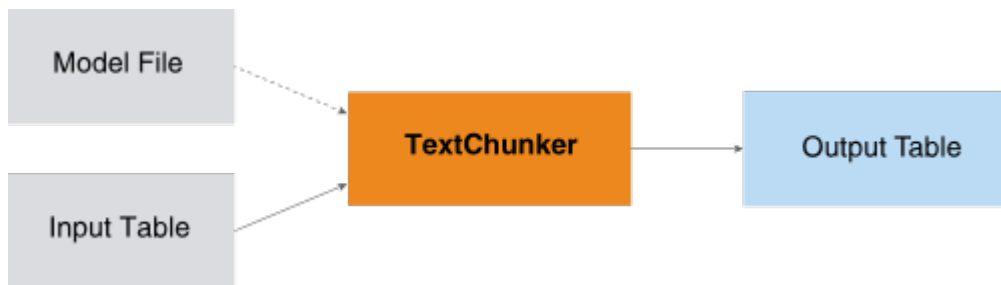
[illegible]

[illegible]

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

### TextChunker (ML Engine)

The `TextChunker` function divides text into phrases and assigns each phrase a tag that identifies its type.



*Text chunking* (also called *shallow parsing*) divides text into phrases in such a way that syntactically related words become members of the same phrase. Phrases do not overlap; that is, a word is a member of only one chunk.

For example, the sentence "He reckons the current account deficit will narrow to only # 1.8 billion in September ." can be divided as follows, with brackets delimiting phrases:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September]

After each opening bracket is a tag that identifies the chunk type (NP, VP, and so on). For information about chunk types, see TextChunker Output.

For more information about text chunking, see:

- Erik F. Tjong Kim Sang and Sabine Buchholz, Introduction to the CoNLL-2000 Shared Task: Chunking. In: *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.
- Fei Sha and Fernando Pereira, Shallow Parsing with Conditional Random Fields. [2003]

TextChunker uses files that are preinstalled on ML Engine. For details, see Preinstalled Files That Functions Use.

## TextChunker Syntax

### Version 1.6

```

SELECT * FROM TextChunker (
  ON { table | view | (query) } PARTITION BY partition_key ORDER BY word_sn
  USING
  WordColumn ('word_column')
  POSColumn ('pos_tag_column')
) AS alias;
  
```

The input\_table is output table of the POSTagger (ML Engine) function, which contains the columns partition\_key and word\_sn.

## TextChunker Syntax Elements

WordColumn

Specify the name of the input table column that contains the words to chunk into phrases. Typically, this is the word column of the output table of the POSTagger function (described in POSTagger Output).

#### POSColumn

Specify the name of the input table column the part-of-speech (POS) tag of words. Typically, this is the pos\_tag column of the output table of the POSTagger function (described in "POSTagger Output").

## TextChunker Input

Table	Description
Input table	POSTagger Output table.  When running POSTagger to create this table, specify in the Accumulate syntax element the name of the input column that contains the unique row identifiers.
Model file	chunker_default_model.bin, provided with function.

## TextChunker Output

### Output Table Schema

Column	Data Type	Description
partition_key	VARCHAR	Key of partition that contains text.
chunk_sn	INTEGER	Sequence number of phrase in sentence.
chunk	VARCHAR	Text chunk (syntactically related words).
chunk_tag	VARCHAR	Phrase type tag (see following table).

## Phrase Type Tags

Tag	Phrase Type
NP	noun phrase
VP	verb phrase
PP	prepositional phrase
ADVP	adverb phrase
SBAR	subordinated clause
ADJP	adjective phrase
PRT	particles

Tag	Phrase Type
CONJP	conjunction phrase
INTJ	interjection
LST	list marker
UCP	unlike coordinated phrase
O	punctuation marks

## TextChunker Examples

### TextChunker Example: POSTagger Output as Input

#### Input

- Input table: pos\_tmp, created by inputting the table cities to the POSTagger function

cities

paraid	paratext
1	I live in Los Angeles.
2	New York is a great city.
3	Chicago is a lot of fun, but the winters are very cold and windy.
4	Philadelphia and Boston have many historical sites.

This statement creates pos\_tmp:

```
CREATE multiset table pos_tmp AS (
  SELECT * FROM POSTagger (
    ON cities
    USING
    Accumulate ('paraid')
    TextColumn ('paratext')
  ) AS dt1
) WITH DATA;
```

#### SQL Call

```
SELECT * FROM TextChunker (
  ON pos_tmp PARTITION BY paraid ORDER BY paraid, word_sn
  USING
  WordColumn ('word')
  POSColumn ('pos_tag')
) AS dt;
```

## Output

partition_key	chunk_sn	chunk	chunk_tag
1	1	i	NP
1	2	live	VP
1	3	in	PP
1	4	los angeles	NP
1	5	.	O
2	1	new york	NP
2	2	is	VP
2	3	a great city	NP
2	4	, filled	VP
2	5	with	PP
2	6	arts and culture	NP
2	7	.	O
3	1	chicago	NP
3	2	is	VP
3	3	a lot	NP
3	4	of	PP
3	5	fun	NP
3	6	,	O
3	7	but	O
3	8	the winters	NP
3	9	are	VP
3	10	very cold and windy	NP
3	11	.	O
4	1	philadelphia and boston	NP
4	2	have	VP
4	3	many historical sites	NP
4	4	.	O

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextChunker Example: SentenceExtractor and POSTagger Output as Input

### Input

paragraphs\_input

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of n points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible.

paraid	paratopic	paratext
3	Logistic Regression	<p>Logistic regression was developed by statistician David Cox in 1958[2][3] (although much work was done in the single independent variable case almost two decades earlier). The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). As such it is not a classification method. It could be called a qualitative response/discrete choice model in the terminology of economics.</p>
4	Cluster analysis	<p>Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. Cluster analysis itself is not one specific algorithm, but the general task to solve. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them.</p>

paraid	paratopic	paratext
5	Association rule learning	Association rule learning is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Rakesh Agrawal et al.[2] introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule {onions, potatoes} => {burger} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat.

## SQL Call

TextChunker requires each sentence to have a unique identifier, and the input to TextChunker must be partitioned by that identifier.

```

SELECT * FROM TextChunker (
  ON (
    SELECT * FROM POSTagger (
      ON (
        SELECT paraid*1000+sentence_sn AS sentence_id, sentence FROM SentenceExtractor (
          ON paragraphs_input
          USING
            TextColumn ('paratext')
            Accumulate ('paraid')
        ) AS dt1
      )
      USING
        TextColumn ('sentence')
        Accumulate ('sentence_id')
    ) AS dt2
  ) PARTITION BY sentence_id ORDER BY word_sn
  USING
    WordColumn('word')

```

```
POSColumn('pos_tag')
) AS dt;
```

## Output

```
partition_key chunk_sn chunk
```

```
-----
1001      1 decision tree learning
1001      2 uses
1001      3 a decision tree
1001      4 as
1001      5 a predictive model
1001      6 which
1001      7 maps
1001      8 observations
1001      9 about
1001     10 an item
1001     11 to
1001     12 conclusions
1001     13 about
1001     14 the items target value
1001     15 .
1001     16 it
1001     17 is
1001     18 one
1001     19 of
1001     20 the predictive modelling approaches
1001     21 used
1001     22 in
1001     23 statistics , data mining and machine learning . tree models
1001     24 where
1001     25 the target variable
1001     26 can take
1001     27 a finite set
1001     28 of
1001     29 values
1001     30 are called
1001     31 classification trees
1001     32 .
1001     33 in
1001     34 these tree structures
1001     35 ,
1001     36 leaves
1001     37 represent class labels and branches
1001     38 represent
1001     39 conjunctions
1001     40 of
1001     41 features
```

1001 42 that  
1001 43 lead  
1001 44 to  
1001 45 those class labels . decision trees  
1001 46 where  
1001 47 the target variable  
1001 48 can take  
1001 49 continuous values  
1001 50 ( typically real numbers  
1001 51 )  
1001 52 are called  
1001 53 regression trees  
1001 54 .  
2001 1 in  
2001 2 statistics  
2001 3 ,  
2001 4 simple linear regression  
2001 5 is  
2001 6 the least squares estimator  
2001 7 of  
2001 8 a linear regression model  
2001 9 with  
2001 10 a single explanatory variable .  
2001 11 in  
2001 12 other words  
2001 13 ,  
2001 14 simple linear regression  
2001 15 fits  
2001 16 a straight line  
2001 17 through  
2001 18 the set  
2001 19 of  
2001 20 n points  
2001 21 in  
2001 22 such a way  
2001 23 that  
2001 24 makes  
2001 25 the sum  
2001 26 of  
2001 27 squared residuals  
2001 28 of  
2001 29 the model (  
2001 30 that  
2001 31 is  
2001 32 , vertical distances  
2001 33 between  
2001 34 the points  
2001 35 of

2001 36 the data  
 2001 37 set  
 2001 38 and  
 2001 39 the fitted line  
 2001 40 )  
 2001 41 as small  
 2001 42 as  
 2001 43 possible  
 2001 44 .  
 3001 1 logistic regression  
 3001 2 was developed  
 3001 3 by  
 3001 4 statistician david cox  
 3001 5 in  
 3001 6 1958[2][3](although much work  
 3001 7 was done  
 3001 8 in  
 3001 9 the single independent variable case  
 3001 10 almost  
 3001 11 two decades  
 3001 12 earlier)  
 3001 13 .  
 3001 14 the binary logistic model  
 3001 15 is used to estimate  
 3001 16 the probability  
 3001 17 of  
 3001 18 a binary response  
 3001 19 based  
 3001 20 on  
 3001 21 one or more predictor ( or independent ) variables ( features) .  
 3001 22 as  
 3001 23 such  
 3001 24 it  
 3001 25 is  
 3001 26 not  
 3001 27 a classification method  
 3001 28 .  
 3001 29 it  
 3001 30 could be called  
 3001 31 a qualitative response/discrete choice model  
 3001 32 in  
 3001 33 the terminology  
 3001 34 of  
 3001 35 economics  
 3001 36 .  
 4001 1 cluster analysis or clustering  
 4001 2 is  
 4001 3 the task

4001 4 of  
4001 5 grouping  
4001 6 a set  
4001 7 of  
4001 8 objects  
4001 9 in  
4001 10 such a way  
4001 11 that  
4001 12 objects  
4001 13 in  
4001 14 the same group  
4001 15 ( called  
4001 16 a cluster )  
4001 17 are  
4001 18 more similar  
4001 19 (  
4001 20 in  
4001 21 some sense  
4001 22 or  
4001 23 another )  
4001 24 to  
4001 25 each other  
4001 26 than  
4001 27 to  
4001 28 those  
4001 29 in  
4001 30 other groups  
4001 31 ( clusters)  
4001 32 .  
4001 33 it  
4001 34 is  
4001 35 a main task  
4001 36 of  
4001 37 exploratory data mining  
4001 38 ,  
4001 39 and  
4001 40 a common technique  
4001 41 for  
4001 42 statistical data analysis  
4001 43 , used  
4001 44 in  
4001 45 many fields  
4001 46 ,  
4001 47 including  
4001 48 machine learning  
4001 49 ,  
4001 50 pattern recognition , image analysis , information retrieval , and bioinformatics  
4001 51 itself

4001 52 is  
4001 53 not  
4001 54 one specific algorithm  
4001 55 ,  
4001 56 but  
4001 57 the general task  
4001 58 to be solved  
4001 59 .  
4001 60 it  
4001 61 can be achieved  
4001 62 by  
4001 63 various algorithms  
4001 64 that  
4001 65 differ  
4001 66 significantly  
4001 67 in  
4001 68 their notion  
4001 69 of  
4001 70 what  
4001 71 constitutes  
4001 72 a cluster  
4001 73 and  
4001 74 how  
4001 75 to efficiently find  
4001 76 them  
4001 77 .  
5001 1 association rule learning  
5001 2 is  
5001 3 a method  
5001 4 for  
5001 5 discovering  
5001 6 interesting relations  
5001 7 between  
5001 8 variables  
5001 9 in  
5001 10 large databases  
5001 11 .  
5001 12 it  
5001 13 is intended to identify  
5001 14 strong rules  
5001 15 discovered  
5001 16 in  
5001 17 databases  
5001 18 using  
5001 19 different measures  
5001 20 of  
5001 21 interestingness  
5001 22 . based

5001 23 on  
 5001 24 the concept  
 5001 25 of  
 5001 26 strong rules  
 5001 27 ,  
 5001 28 rakesh agrawal et al.[2 ] introduced association rules  
 5001 29 for  
 5001 30 discovering regularities  
 5001 31 between  
 5001 32 products  
 5001 33 in  
 5001 34 large-scale transaction data  
 5001 35 recorded  
 5001 36 by  
 5001 37 point-of-sale ( pos ) systems  
 5001 38 in  
 5001 39 supermarkets  
 5001 40 .  
 5001 41 for  
 5001 42 example  
 5001 43 ,  
 5001 44 the rule { onions , potatoes} $\Rightarrow$ {burger  
 5001 45 } found  
 5001 46 in  
 5001 47 the sales data  
 5001 48 of  
 5001 49 a supermarket  
 5001 50 would indicate  
 5001 51 that  
 5001 52 if  
 5001 53 a customer  
 5001 54 buys  
 5001 55 onions  
 5001 56 and  
 5001 57 potatoes  
 5001 58 together  
 5001 59 ,  
 5001 60 they  
 5001 61 are  
 5001 62 likely  
 5001 63 to also buy  
 5001 64 hamburger meat  
 5001 65 .

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextParser (ML Engine)

The TextParser function tokenizes an input stream of words, optionally *stems* them (reduces them to their root forms), and then outputs them. The function can either output all words in one row or output each word in its own row with (optionally) the number of times that the word appears.

The TextParser function uses Porter2 as the stemming algorithm.

The TextParser function reads a document into a memory buffer and creates a hash table. The dictionary for the document must not exceed available memory; however, a million-word dictionary with an average word length of ten bytes requires only 10 MB of memory.

TextParser uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## TextParser Syntax

### Version 1.14

```
SELECT * FROM TextParser (
  ON { table | view | (query) } [ PARTITION BY expression [,...] ]
  USING
  TextColumn ('text_column')
  [ ConvertToLowerCase ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemTokens ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Delimiter ('delimiter_regular_expression') ]
  [ OutputTotalWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Punctuation ('punctuation_regular_expression') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ TokenColName ('token_column') ]
  [ FrequencyColName ('frequency_column') ]
  [ TotalColName ('total_column') ]
  [ RemoveStopWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ PositionColName ('position_column') ]
  [ ListPositions ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ OutputByWord ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemExceptions ('exception_rule_file') ]
  [ StopWordsList ('stop_word_file') ]
) AS alias;
```

If you include the PARTITION BY clause, the function treats all rows in the same partition as a single document. If you omit the PARTITION BY clause, the function treats each row as a single document.

Related information

- Column Specification Syntax Elements
- Regular Expressions in Syntax Elements

## TextParser Syntax Elements

### TextColumn

Specify the name of the input column with contents to tokenize.

### ConvertToLowerCase

[Optional] Specify whether to convert input text to lowercase.

The function ignores this syntax element if the StemTokens syntax element has the value 'true'.

Default: 'true'

### StemTokens

[Optional] Specify whether to stem the tokens—that is, whether to apply the Porter2 stemming algorithm to each token to reduce it to its root form. Before stemming, the function converts the input text to lowercase and applies the RemoveStopWords syntax element.

Default: 'false'

### Delimiter

[Optional] Specify a regular expression that represents the word delimiter.

The function uses only specified characters as delimiters. For example, if you specify Delimiter ('-'), the function uses only the hyphen character as a delimiter. To use the hyphen and the default delimiters, specify Delimiter ('[- \t\f\r\n]+').

Default: '[ \t\f\r\n]+'

### OutputTotalWords

[Optional] Specify whether to output a column that contains the total number of words in the input document.

Default: 'false'

### Punctuation

[Optional] Specify a regular expression that represents the punctuation characters to remove from the input text. With StemTokens ('true'), the recommended value is '[\\[.,?!:;~()\\]]+'.

Default: '[.,!?!:]'

### Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

No accumulate\_column can be the same as token\_column or total\_column.

Default: All input columns

### TokenColName

[Optional] Specify the name of the output column that contains the tokens.

Default: 'token'

### FrequencyColName

[Optional] Specify the name of the output column that contains the frequency of each token.

The function ignores this syntax element if the OutputByWord syntax element has the value 'false'.

Default: 'frequency'

### TotalColName

[Optional] Specify the name of the output column that contains the total number of words in the input document.

Default: 'total\_count'

### RemoveStopWords

[Optional] Specify whether to remove stop words from the input text before parsing.

Default: 'false'

### PositionColName

[Optional] Specify the name of the output column that contains the position of a word within a document.

Default: 'location'

#### ListPositions

[Optional] Specify whether to output the position of a word in list form.

The function ignores this syntax element if the OutputByWord syntax element has the value 'false'.

Default: 'false' (The function outputs a row for each occurrence of the word.)

#### OutputByWord

[Optional] Specify whether to output each token of each input document in its own row in the output table. If you specify 'false', then the function outputs each tokenized input document in one row of the output table.

Default: 'true'

#### StemExceptions

[Optional] Specify the location of the file that contains the stemming exceptions. A stemming exception is a word followed by its stemmed form. The word and its stemmed form are separated by white space. Each stemming exception is on its own line in the file. For example:

```
bias bias
news news
goods goods
lying lie
ugly ugli
sky sky
early earli
```

The words 'lying', 'ugly', and 'early' are to become 'lie', 'ugli', and 'earli', respectively. The other words are not to change.

Default: No stemming exceptions

#### StopWordsList

[Optional] Specify the location of the file that contains the stop words (words to ignore when parsing text). Each stop word is on its own line in the file. For example:

```
a
an
the
and
this
with
but
will
```

Default: No stop words

## TextParser Input

If you include the PARTITION BY clause, the function treats all rows in the same partition as a single document. If you omit the PARTITION BY clause, the function treats each row as a single document.

## Input Table Schema

Column	Data Type	Description
text_column	VARCHAR	Text to parse.
accumulate_column	Any	[Column appears once for each specified accumulate_column.] Column to copy to output table.

## TextParser Output

The output table schema depends on the OutputByWord syntax element.

### Output Table Schema, Output\_By\_Word ('true') (Default)

Column	Data Type	Description
accumulate_column	Same as in input table	[Column appears once for each specified accumulate_column.] Column copied from input table.
token_column	CLOB	Token.
frequency_column	INTEGER	Frequency of token.
position_column	VARCHAR	Position of word within document.

### Output Table Schema, Output\_By\_Word ('false')

Column	Data Type	Description
accumulate_column	Same as in input table	[Column appears once for each specified accumulate_column.] Column copied from input table.
token_column	CLOB	Token.

## TextParser Examples

### TextParser Example: StopWordsList, No StemExceptions

#### Input

- InputTable: complaints, a log of vehicle complaints.

The category column indicates whether the vehicle was in a crash.

- Stop words file: stopwords.txt, which is preinstalled on ML Engine (shown in TextClassifierTrainer Example)

complaints

doc_id	text_data	category
1	consumer was driving approximately 45 mph hit a deer with the front bumper and then ran into an embankment head-on passenger's side air bag did deploy hit windshield and deployed outward. driver's side airbag cover opened but did not inflate it was still folded causing injuries.	crash
2	when vehicle was involved in a crash totalling vehicle driver's side/ passenger's side air bags did not deploy. vehicle was making a left turn and was hit by a ford f350 traveling about 35 mph on the front passenger's side. driver hit his head-on the steering wheel. hurt his knee and received neck and back injuries.	crash
3	consumer has experienced following problems; 1.) both lower ball joints wear out excessively; 2.) head gasket leaks; and 3.) cruise control would shut itself off while driving without foot pressing on brake pedal.	no_crash
...	...	...

## SQL Call

```

SELECT * FROM TextParser (
  ON complaints
  USING
  TextColumn ('text_data')
  ConvertToLowerCase ('true')
  StemTokens ('false')
  OutputByWord ('true')
  Punctuation ('\[.,?!\\]')
  RemoveStopWords ('true')
  ListPositions ('true')
  Accumulate ('doc_id', 'category')
  StopWordsList ('stopwords.txt')
) AS dt ORDER BY doc_id,category,token,frequency,location;

```

## Output

doc_id	category	token	frequency	location
1	crash	45	1	4
1	crash	air	1	22
1	crash	airbag	1	33
1	crash	approximately	1	3
1	crash	bag	1	23
1	crash	bumper	1	12
1	crash	causing	1	44
1	crash	consumer	1	0
1	crash	cover	1	34
1	crash	deer	1	8
1	crash	deploy	1	25
1	crash	deployed	1	29
1	crash	did	2	24,37
1	crash	driver's	1	31
1	crash	driving	1	2
1	crash	embankment	1	18
1	crash	folded	1	43
1	crash	front	1	11
1	crash	head-on	1	19
1	crash	hit	2	6,26
1	crash	inflate	1	39
1	crash	injuries	1	45
1	crash	it	1	40
1	crash	mph	1	5
1	crash	not	1	38
1	crash	opened	1	35
1	crash	outward	1	30
1	crash	passenger's	1	20
1	crash	ran	1	15
1	crash	side	2	21,32
1	crash	still	1	42
1	crash	then	1	14
1	crash	windshield	1	27
2	crash	35	1	33
2	crash	about	1	32
2	crash	air	1	13
2	crash	back	1	54
2	crash	bags	1	14
2	crash	by	1	27
2	crash	crash	1	6
2	crash	deploy	1	17
2	crash	did	1	15
2	crash	driver	1	40

2 crash	driver's	1 9
2 crash	f350	1 30
2 crash	ford	1 29
2 crash	front	1 37
2 crash	head-on	1 43
2 crash	his	2 42,48
2 crash	hit	2 26,41
2 crash	hurt	1 47
2 crash	injuries	1 55
2 crash	involved	1 3
2 crash	knee	1 49
2 crash	left	1 22
2 crash	making	1 20
2 crash	mph	1 34
2 crash	neck	1 52
2 crash	not	1 16
2 crash	on	1 35
2 crash	passenger's	2 11,38
2 crash	received	1 51
2 crash	side	2 12,39
2 crash	side/	1 10
2 crash	steering	1 45
2 crash	totalling	1 7
2 crash	traveling	1 31
2 crash	turn	1 23
2 crash	vehicle	3 1,8,18
2 crash	wheel	1 46
2 crash	when	1 0
3 no_crash	1)	1 5
3 no_crash	2)	1 13
3 no_crash	3)	1 18
3 no_crash	ball	1 8
3 no_crash	both	1 6
3 no_crash	brake	1 31
3 no_crash	consumer	1 0
3 no_crash	control	1 20
3 no_crash	cruise	1 19
3 no_crash	driving	1 26
3 no_crash	excessively;	1 12
3 no_crash	experienced	1 2
3 no_crash	following	1 3
3 no_crash	foot	1 28
3 no_crash	gasket	1 15
3 no_crash	has	1 1
3 no_crash	head	1 14
3 no_crash	itself	1 23
3 no_crash	joints	1 9
3 no_crash	leaks;	1 16

3 no_crash lower	1 7
3 no_crash off	1 24
3 no_crash on	1 30
3 no_crash out	1 11
3 no_crash pedal	1 32
3 no_crash pressing	1 29
3 no_crash problems;	1 4
3 no_crash shut	1 22
3 no_crash wear	1 10
3 no_crash while	1 25
3 no_crash without	1 27
3 no_crash would	1 21
4 no_crash after	1 6
4 no_crash back	1 18
4 no_crash been	1 40
4 no_crash case	2 1,36
4 no_crash completed	1 10
4 no_crash consumer	1 15
4 no_crash dealer	2 20,22
4 no_crash driveshaft	1 31
4 no_crash has	1 39
4 no_crash heard	1 13
4 no_crash hitting	1 33
4 no_crash informed	1 26
4 no_crash intermittently	1 14
4 no_crash manufacturer	1 38
4 no_crash noise	1 11
4 no_crash notified	1 41
4 no_crash owner	1 28
4 no_crash recall	1 5
4 no_crash reinspected	1 23
4 no_crash repaired	1 3
4 no_crash that	1 29
4 no_crash took	1 16
4 no_crash transfer	2 0,35
4 no_crash under	1 4
4 no_crash vehicle	2 17,24
4 no_crash work	1 8
5 no_crash &	2 21,27
5 no_crash 10mph	1 8
5 no_crash 3	1 14
5 no_crash accurate	1 41
5 no_crash almost	1 33
5 no_crash also	1 36
5 no_crash at	1 19
5 no_crash be	1 12
5 no_crash blew	1 34
5 no_crash by	1 56

5 no_crash checked	1 18
5 no_crash dealership	1 20
5 no_crash defect	1 32
5 no_crash does	1 38
5 no_crash factory	1 31
5 no_crash fail	1 48
5 no_crash had	1 16
5 no_crash if	1 43
5 no_crash increasedit	1 46
5 no_crash informed	1 23
5 no_crash it's	1 29
5 no_crash just	1 7
5 no_crash keep	1 40
5 no_crash manufacturer	1 57
5 no_crash mechanic	1 55
5 no_crash not	1 39
5 no_crash over	1 13
5 no_crash referred	1 53
5 no_crash rpms	1 10
5 no_crash slip	1 4
5 no_crash speed	1 44
5 no_crash speedometer	1 37
5 no_crash speeds	1 42
5 no_crash start	1 2
5 no_crash stuck	1 26
5 no_crash that	1 28
5 no_crash thousand	1 15
5 no_crash transmission	2 0,24
5 no_crash traveling	1 6
5 no_crash up	1 35
5 no_crash vehicle	1 17
5 no_crash when	1 5
5 no_crash work	1 50
5 no_crash would	3 1,11,47
6 no_crash also	1 21
6 no_crash belts/speed	1 27
6 no_crash burned	1 7
6 no_crash cable	1 5
6 no_crash coil	1 9
6 no_crash controlcable	1 28
6 no_crash could	1 15
6 no_crash crash	1 20
6 no_crash dealer	1 22
6 no_crash defective	1 3
6 no_crash drive	1 26
6 no_crash due	1 0
6 no_crash further	1 36
6 no_crash have	1 16

6 no_crash ignition	1 4
6 no_crash information	1 37
6 no_crash performed	1 30
6 no_crash please	1 34
6 no_crash provide	1 35
6 no_crash r&r	1 25
6 no_crash replaced	1 23
6 no_crash resulted	1 17
6 no_crash stalled	1 12
6 no_crash tune	1 32
6 no_crash unexpectedly	1 13
6 no_crash up	1 33
6 no_crash vehicle	2 11,31
6 no_crash which	2 6,14
7 no_crash &	1 16
7 no_crash 97v017000	1 28
7 no_crash by	1 25
7 no_crash do	1 22
7 no_crash have	1 12
7 no_crash jiggle	1 14
7 no_crash move	1 20
7 no_crash not	1 8
7 no_crash off/on	1 24
7 no_crash on	1 4
7 no_crash properly	1 10
7 no_crash recall	1 27
7 no_crash switch	2 1,15
7 no_crash themselves	1 26
7 no_crash then	1 17
7 no_crash turn	1 23
7 no_crash turned	1 3
7 no_crash when	1 0
7 no_crash windshield	1 5
7 no_crash wipers	3 6,18,21
7 no_crash work	1 9
7 no_crash would	3 7,11,19
8 no_crash consumer	1 0
8 no_crash driving	1 2
8 no_crash happened	1 13
8 no_crash periodcally	1 14
8 no_crash rain	1 5
8 no_crash stopped	1 11
8 no_crash storm	1 6
8 no_crash when	1 7
8 no_crash windshield	1 9
8 no_crash wipers	1 10
9 no_crash *ml	1 21
9 no_crash 66900	1 1

9 no_crash at	2 0,16
9 no_crash expense	1 18
9 no_crash first	1 11
9 no_crash gear	1 12
9 no_crash has	1 4
9 no_crash made	1 15
9 no_crash malfunctioned	1 5
9 no_crash miles	1 2
9 no_crash not	1 8
9 no_crash owner's	1 17
9 no_crash reimbursement	1 20
9 no_crash repairs	1 13
9 no_crash shift	1 9
9 no_crash transmission	1 3
9 no_crash wants	1 19
9 no_crash were	1 14
10 no_crash 1998	1 33
10 no_crash aware	1 14
10 no_crash been	1 21
10 no_crash by	1 27
10 no_crash corrected	1 22
10 no_crash has	1 19
10 no_crash hill	1 29
10 no_crash incline	1 6
10 no_crash it	1 7
10 no_crash its	1 10
10 no_crash manufactured	1 31
10 no_crash manufacturer	1 12
10 no_crash not	1 20
10 no_crash of	1 15
10 no_crash on	2 4,9
10 no_crash own	1 11
10 no_crash owned	1 26
10 no_crash problem	2 17,18
10 no_crash recker	1 30
10 no_crash rolled	1 8
10 no_crash sitting	1 3
10 no_crash truck	2 1,24
10 no_crash walnut	1 28
10 no_crash when	1 0
11 crash approximately	1 23
11 crash been	1 20
11 crash building	1 17
11 crash car	3 0,7,18
11 crash condition	1 32
11 crash crashed	1 11
11 crash engine	1 1
11 crash fence	1 14

11 crash	for	1 29
11 crash	forward	1 9
11 crash	had	1 19
11 crash	high	1 30
11 crash	idle	1 31
11 crash	incident	1 28
11 crash	lurched	1 8
11 crash	one	1 24
11 crash	park	1 6
11 crash	prior	1 26
11 crash	raced	1 2
11 crash	shop	1 22
11 crash	slowing	1 4
11 crash	week	1 25
11 crash	while	1 3
12 crash	65	1 5
12 crash	70mph	1 7
12 crash	airbags	1 15
12 crash	another	1 2
12 crash	at	1 4
12 crash	dealer	1 17
12 crash	deployed	1 16
12 crash	driver's	1 10
12 crash	ended	1 1
12 crash	has	1 18
12 crash	neither	1 9
12 crash	or	1 12
12 crash	passenger's	1 13
12 crash	rear	1 0
12 crash	side	2 11,14
12 crash	vehicle	2 3,19
13 no_crash	around	1 27
13 no_crash	coming	1 25
13 no_crash	compartment	1 17
13 no_crash	drivers	1 28
13 no_crash	ea02-025	1 34
13 no_crash	engine	1 16
13 no_crash	fire	2 8,24
13 no_crash	for	1 4
13 no_crash	from	1 26
13 no_crash	front	1 30
13 no_crash	hour	1 6
13 no_crash	left	1 12
13 no_crash	of	1 14
13 no_crash	on	1 10
13 no_crash	owner	1 22
13 no_crash	owners	1 18
13 no_crash	parked	1 3

13 no_crash referenced	1 32
13 no_crash saw	1 23
13 no_crash side	2 13,29
13 no_crash smelled	1 20
13 no_crash smoke	1 21
13 no_crash son	1 19
13 no_crash started	1 9
13 no_crash vehicle	1 1
13 no_crash wheel	1 31
13 no_crash while	1 0
14 no_crash	1 14
14 no_crash 99v029000	1 6
14 no_crash after	1 0
14 no_crash airbag	1 10
14 no_crash been	1 21
14 no_crash dealer	1 16
14 no_crash has	1 20
14 no_crash ignition	1 7
14 no_crash light	1 11
14 no_crash manufacturer	1 19
14 no_crash notified	1 22
14 no_crash on	1 13
14 no_crash recall	1 5
14 no_crash repaired	1 3
14 no_crash stayed	1 12
14 no_crash switch	1 8
14 no_crash under	1 4
14 no_crash vehicle	1 1
15 no_crash 4	1 27
15 no_crash alternator/	1 20
15 no_crash battery	1 21
15 no_crash become	1 13
15 no_crash cannot	1 33
15 no_crash causing	2 6,37
15 no_crash change	1 19
15 no_crash consumer	1 16
15 no_crash control	1 1
15 no_crash defect	1 30
15 no_crash determine	1 34
15 no_crash electrical	1 0
15 no_crash engine	1 11
15 no_crash had	1 17
15 no_crash inoperative	1 15
15 no_crash module	2 2,25
15 no_crash occurring	1 32
15 no_crash out	1 5
15 no_crash problem	1 39
15 no_crash replaced	1 26

15 no_crash shortening	1 4
15 no_crash stall	1 10
15 no_crash starter	1 23
15 no_crash still	1 31
15 no_crash times	1 28
15 no_crash totally	1 14
15 no_crash vehicle	1 8
15 no_crash what	1 35
16 no_crash 68000	1 1
16 no_crash also	1 17
16 no_crash at	1 0
16 no_crash broke	1 5
16 no_crash caused	1 18
16 no_crash causing	1 10
16 no_crash down	1 23
16 no_crash housing	1 8
16 no_crash loss	1 12
16 no_crash miles	1 2
16 no_crash of	1 13
16 no_crash off	1 6
16 no_crash power	2 3,14
16 no_crash pump	1 9
16 no_crash shut	1 22
16 no_crash steering	2 4,15
16 no_crash total	1 11
16 no_crash vehicle	1 20
16 no_crash which	1 16
17 crash 50	1 14
17 crash 80	1 17
17 crash air	2 25,37
17 crash airbags	1 4
17 crash another	1 10
17 crash approximately	1 13
17 crash at	2 12,16
17 crash bags	2 26,38
17 crash consumer	1 8
17 crash deploy	3 7,29,41
17 crash determine	1 35
17 crash did	4 5,27,33,39
17 crash driver	1 30
17 crash dual	1 3
17 crash head-on	1 22
17 crash hit	1 19
17 crash impact	1 24
17 crash injuriesdealer	1 32
17 crash mph	1 18
17 crash mphand	1 15
17 crash not	4 6,28,34,40

17	crash	occasions	1 2
17	crash	on	1 0
17	crash	rearended	1 9
17	crash	sustained	1 31
17	crash	truck	1 21
17	crash	two	1 1
17	crash	upon	1 23
17	crash	vehicle	1 11
17	crash	why	1 36
18	no_crash	leaking	1 2
18	no_crash	sunroof	1 0
18	no_crash	yh	1 3
19	no_crash	be	1 9
19	no_crash	frame	1 3
19	no_crash	from	1 5
19	no_crash	manufacturer	1 7
19	no_crash	motor	1 0
19	no_crash	notified	1 10
19	no_crash	separated	1 4
19	no_crash	vehicle	1 6
20	no_crash	about	1 19
20	no_crash	bearing	1 3
20	no_crash	brake's	1 17
20	no_crash	broke	1 4
20	no_crash	can't	1 25
20	no_crash	causing	1 5
20	no_crash	consumer	1 15
20	no_crash	dealer	1 24
20	no_crash	determine	1 26
20	no_crash	down	1 14
20	no_crash	four	1 20
20	no_crash	front	1 1
20	no_crash	had	1 16
20	no_crash	left	1 11
20	no_crash	problem	1 28
20	no_crash	pull	1 8
20	no_crash	rear	1 0
20	no_crash	replaced	1 18
20	no_crash	slowing	1 13
20	no_crash	still	1 23
20	no_crash	times	1 21
20	no_crash	vehicle	1 6
20	no_crash	wheel	1 2
20	no_crash	when	1 12

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextParser Example: StemExceptions, No StopWordsList

### Input

- Input table: complaints\_mini, which is the first two rows of complaints, the TextParser Example: StopWordsList, No StemExceptions InputTable

complaints\_mini

doc_id	text_data	category
1	consumer was driving approximately 45 mph hit a deer with the front bumper and then ran into an enbankment head-on passenger's side air bag did deploy hit windshield and deployed outward. driver's side airbag cover opened but did not inflate it was still folded causing injuries.	crash
2	when vehicle was involved in a crash totalling vehicle driver's side/ passenger's side air bags did not deploy. vehicle was making a left turn and was hit by a ford f350 traveling about 35 mph on the front passenger's side. driver hit his head-on the steering wheel. hurt his knee and received neck and back injuries.	crash

The stemming exceptions table, stemmingexception.text, contains:

```
consumer customer
enbankment embankment
```

### SQL Call

```
SELECT * FROM TextParser (
  ON complaints_mini
  USING
  TextColumn ('text_data')
  ConvertToLowerCase ('true')
  StemTokens ('true')
  OutputByWord ('false')
  Punctuation ('\[,.\?!\\')
  Accumulate ('doc_id', 'category')
  StemExceptions ('stemmingexception.txt')
) AS dt ORDER BY doc_id;
```

## Output

```
doc_id category tokens
```

```
-----
1 crash      customer was drive approxim 45 mph hit a deer with the front bumper and then ran into
2 crash      when vehicl was involv in a crash total vehicl driver side/ passeng side air bag did
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextTagger (ML Engine)

The TextTagger function tags text documents according to user-defined rules that use text-processing and logical operators. The text in the documents can include Unicode emoticons (also called emojis).

You can run queries with emojis only from the bteq prompt, not using Teradata Studio™.

## TextTagger Syntax

### Version 1.7

```
SELECT * FROM TextTagger (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON { table | view | (query) } AS Rules DIMENSION ]
  USING
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ TaggingRules ('rule AS tag' [...]) ]
  [ Tokenize ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ OutputByTag ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ TagDelimiter ('delimiter') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range } [...]) ]
) AS alias;
```

Related information

Related information

Column Specification Syntax Elements

## TextTagger Syntax Elements

InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**TaggingRules**

[Required if you do not specify a Rules table, disallowed otherwise.] Specify the tag names and tagging rules. For information about defining tagging rules, see [Defining Tagging Rules](#).

**Tokenize**

[Optional] Specify whether the function tokenizes the input text before evaluating the rules and tokenizes the text string parameter in the rule definition when parsing a rule.

If you specify 'true', then you must also specify the InputLanguage syntax element. The function uses the value of InputLanguage to create the word tokenizer.

Default: 'false'

**OutputByTag**

[Optional] Specify whether the function outputs a tuple when a text document matches multiple tags.

Default: 'false' (One tuple in the output stands for one document and the matched tags are listed in the output column tag.)

**TagDelimiter**

[Optional]

Specify the delimiter, a string, that separates multiple tags in the output column tag if OutputByTag has the value 'false'. If OutputByTag has the value 'true', specifying this syntax element causes an error.

Default: ',' (comma)

**Accumulate**

[Optional] Specify the names of text table columns to copy to the output table.

Do not use the name 'tag' for an accumulate\_column, because the function uses that name for the output table column that contains the tags.

## Defining Tagging Rules

You can specify tagging rules with either the TaggingRules syntax element or a Rules table.

## Rules for Rule Operations Table

- The operand opn (where n is 1, 2, or 3) can be any of the following:

opn	Rules for opn
String literal	<ul style="list-style-type: none"> <li>Enclose string literal in double quotation marks (for example, "Start countdown").</li> <li>If string literal contains double quotation marks, precede each double quotation mark with two backslashes (for example, "\"Start countdown\").</li> <li>Do not use the empty string (").</li> <li>If an operation has only string literal operands, matches are case-insensitive and do not consider overlapping.</li> </ul>
Java regular expression (regex"exp")	<ul style="list-style-type: none"> <li>An operation with one or more Java regular expression operands uses fuzzy matching. Fuzzy matching evaluates original text input; that is, matching is case-sensitive and text is not tokenized.</li> </ul>
[superdist operation only] List of string literals or Java regular expressions	<ul style="list-style-type: none"> <li>For details, see description of superdist operation in following table.</li> </ul>

- The operands lower and upper are nonnegative integers.

You can omit either lower or upper, but not both. For example, all of the following are valid syntax for the contain operation:

```
contain (col, op1, lower, upper)
contain (col, op1, lower,)
contain (col, op1,, upper)
```

If x is the number of times that op1 appears in col, then the preceding operations have the following meanings, respectively:

lower <= x <= upper

lower <= x

x <= upper

The meanings of lower, x, and upper depend on the operation.

## Rule Operations

This table summarizes the operations that a rule can use. For simplicity, the table shows only the syntax that specifies both lower and upper.

Syntax	Description
equal (col, op1)	Returns 'true' if the text in column col and the value of op1 are equal; 'false' otherwise.

Syntax	Description
<code>contain (col, op1, lower, upper)</code>	Returns 'true' if, in column <code>col</code> , the number of times that the value of <code>op1</code> appears is in the range <code>[lower, upper]</code> ; 'false' otherwise.
<code>dist (col, op1, op2, lower, upper)</code>	<p>Returns 'true' if, in column <code>col</code>, the distance between the values of <code>op1</code> and <code>op2</code> (that is, the number of words between them) is in the range <code>[lower, upper]</code>; 'false' otherwise.</p> <p>The distance computation depends on the <code>InputLanguage</code> and <code>UseTokenizer</code> syntax elements.</p> <p>By default, <code>InputLanguage</code> is 'en' (English) and <code>UseTokenizer</code> is 'false', and words are delimited by whitespace characters.</p> <p>If <code>InputLanguage</code> is 'zh_cn' (Simplified Chinese) or 'zh_tw' (Traditional Chinese) and <code>UseTokenizer</code> is 'true', then the function performs word segmentation before computing the distance between words.</p>

`superdist (col, op1, op2, con1, op3, con2)`

Returns 'true' if, in column `col`, the values of `op1`, `op2`, and `op3` satisfy the context rules `con1` and `con2`; 'false' otherwise.

The rules `con1` and `con2` specify the context for inclusion and exclusion, as the following table shows.

con1 or con2 Value	con1 Meaning	con2 Meaning
nwn	op2 appears n or fewer words before or after op1.	op3 does not appear n or fewer words before or after op1.
nrn	op2 appears n or fewer words after op1.	op3 does not appear n or fewer words after op1.
para	op2 appears in the same paragraph as op1.	op3 does not appear in the same paragraph as op1.
sent	op2 appears in the same sentence as op1.	op3 does not appear in the same sentence as op1.

The distance computation depends on the `InputLanguage` and `UseTokenizer` syntax elements (for details, see the description of the `dist` operation).

A paragraph ends with either `"\n"` or `"\r\n"`. A sentence ends with either period (`.`), question mark (`?`), or exclamation mark (`!`). The function fragments the input into paragraphs or sentences and then checks the context rule on each piece of text. If one piece satisfies the rule, then the function tags the whole input.

`opn` (where `n` is 1, 2, or 3) can be a list of words. Enclose the list in double quotation marks and separate the words with semicolons. For example: `"good;bad;neutral"`

If `opn` is a Java regular expression, then `exp` can be a list. Separate the items with semicolons. For example:  
`regex"invest[\w]*;volatil[\w]*;risk"`

When a list appears in an inclusion context, the rule is satisfied if at least one item appears in the context. When a list appears in an exclusion context, the rule is satisfied if no item appears in the context.

The operand-context pairs after `op1` are optional; that is, the following are valid syntax:

```
superdist(col, op1,,,)
```

```
superdist(col, op1, op2, con1,,)
```

```
superdist(col, op1,,, op3, con2)
```

```
superdist(col, op1, op2, con1, op3, con2)
```

```
superdist(col, op1,,,)
```

The final syntax in the preceding list returns 'true' if `op1` appears in `col`.

Syntax	Description
dict (col, "[schema/]dictionary",lower, upper)	Returns 'true' if, in column col, the number of items (lines in the dictionary file) is in the range [lower, upper]; 'false' otherwise.  This operation requires that dictionary file [schema.] dictionary is installed on ML Engine. Dictionary name, dictionary, is case-sensitive. If dictionary is in public schema, you can omit schema name, schema.
operation1 and operation2	Returns 'true' if both operation1 and operation2 return 'true'; 'false' otherwise.
operation1 or operation2	Returns 'true' if one or both operation1 or operation2 returns 'true'; 'false' otherwise.
not operation	Returns 'true' if operation returns 'false'; 'false' if operation returns 'true'.

## TextTagger Input

Table	Description
Input table	Contains text to tag.
Rules	[Optional] Contains tagging rules. If you omit this table, specify tagging rules with TaggingRules syntax element.

## InputTable Schema

The table can have additional columns, but the function ignores them unless you specify them in rules.

Column	Data Type	Description
text_column	VARCHAR	Text to tag.
accumulate_column	Any	[Column appears once for each specified accumulate_column.] Column to copy to output table.

## Rules Schema

Column	Data Type	Description
tagname	VARCHAR	Name of tag.
definition	VARCHAR	Definition of tag.

## TextTagger Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as in InputTable	[Column appears once for each specified accumulate_column.] Column copied from InputTable. Typically, one accumulate_column contains document identifiers.
tag	VARCHAR	Tuple of tags that match text document. Tag names come from either TaggingRules syntax element or rules table. If text document matches no tag, its value in this column is an empty string.

### TextTagger Examples

#### TextTagger Example: TaggingRules

#### Input

text\_inputs

id	title	content	catalog
1	Chennai Floods	Chennai floods have battered the capital city of Tamil Nadu and its adjoining areas. Normal life came to a standstill when roads were submerged in water and all modes of transport were severely affected. In the past, Chennai has had tsunamis and earthquakes	Regional
2	Tennis Superstars	Roger Federer born on 8 August 1981, is a greatest tennis player, who has been continuously ranked inside the top 10 since October 2002 and has won Wimbledon, USOpen, Australian and FrenchOpen titles mutiple times	sports

id	title	content	catalog
3	Sports Rivalry	The Federer Nadal rivalry, known by many as Fedal, is between two professional tennis players, Roger Federer of Switzerland and Rafael Nadal of Spain. They are currently engaged in a storied rivalry, which many consider to be the greatest in tennis history. They have played 34 times, most recently in the 2015 Swiss Indoors final, and Nadal leads their eleven-year-old rivalry with an overall record of 231	sports
4	Sports Rivalry	The India Pakistan cricket rivalry is one of the most intense sports rivalries in the world. An India-Pakistan cricket match has been estimated to attract up to one billion viewers, according to TV ratings firms and various other reports. The 2011 World Cup semifinal between the two teams attracted around 988 million television viewers	sports

id	title	content	catalog
5	Sports Rivalry	An Ashes series is traditionally of five Tests, hosted in turn by England and Australia at least once every four years. As of August 2015, England hold the ashes, having won three of the five Tests in the 2015 Ashes series. Overall, Australia has won 32 series, England 32 and five series have been drawn.	sports

## SQL Call

```

SELECT * FROM TextTagger (
  ON text_inputs
  USING
  TaggingRules ('contain(content, "floods", 1,) or
    contain(content, "tsunamis", 1,) AS Natural-Disaster',
    'contain(content, "Roger", 1,) and
    contain(content, "Nadal", 1,) AS Tennis-Rivalry',
    'contain(title, "Tennis", 1,) and
    contain(content, "Roger", 1,) AS Tennis-Greats',
    'contain(content, "India", 1,) and
    contain(content, "Pakistan", 1,) AS Cricket-Rivalry',
    'contain(content, "Australia", 1,) and
    contain(content, "England", 1,) AS The-Ashes'
  )
  OutputByTag ('true')
  Accumulate ('id')
) AS dt ORDER BY id;

```

## Output

```

id tag
-- -----
1 Natural-Disaster
2 Tennis-Greats
3 Tennis-Rivalry
4 Cricket-Rivalry
5 The-Ashes

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextTagger Example: Rules Table

The Rules table, `rule_inputs`, defines the same rules as the `TaggingRules` syntax element in `TextTagger`  
Example: `TaggingRules`.

## Input

- Input table: `text_inputs`, as in `TextTagger Example: TaggingRules`
- Rules: `rule_inputs`

`rule_inputs`

tagname	definition
Cricket-Rivalry	contain(content,"India",1,) and contain(content,"Pakistan",1,)
Natural-Disaster	contain(content, "floods",1,) or contain(content,"tsunamis",1,)
Tennis-Greats	contain(title,"Tennis",1,) and contain(content,"Roger",1,)
Tennis-Rivalry	contain(content,"Roger",1,) and contain(content,"Nadal",1,)
The-Ashes	contain(content,"Australia",1,) and contain(content,"England",1,)

## SQL Call

```
SELECT * FROM TextTagger (
  ON text_inputs PARTITION BY ANY
  ON rule_inputs AS Rules DIMENSION
  USING
    Accumulate ('id')
) AS dt ORDER BY id;
```

## Output

```
id tag
-- -----
1 Natural-Disaster
2 Tennis-Greats
3 Tennis-Rivalry
4 Cricket-Rivalry
5 The-Ashes
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextTagger Example: TaggingRules, Dictionary File

This example uses this dictionary file, keywords.txt:

```
floods
tsunamis
Roger
Nadal
India
Pakistan
England
Australia
```

## Input

- Input table: text\_inputs, as in TextTagger Example: TaggingRules

## SQL Call

```
SELECT * FROM TextTagger (
  ON text_inputs
  USING
  TaggingRules ('dict(content, "keywords.txt", 1,) AND
    equal(titles, "Chennai Floods") AS Natural-Disaster',
    'dict(content, "keywords.txt", 2,) AND
    equal(catalog, "sports") AS Great-Sports-Rivalry '
  )
  Accumulate ('id')
) AS dt ORDER BY id;
```

## Output

```
id tag
-- -----
1 Natural-Disaster
2
3 Great-Sports-Rivalry
4 Great-Sports-Rivalry
5 Great-Sports-Rivalry
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextTagger Example: TaggingRules, Superdist

### Input

- Input table: text\_inputs, as in TextTagger Example: TaggingRules

### SQL Call

```
SELECT * FROM TextTagger (
  ON text_inputs
  USING
  TaggingRules ('superdist(content, "Chennai", "floods", sent, ,)
    AS Chennai-Flood-Disaster',
    'superdist(content, "Roger", "titles", para, "Nadal", para)
    AS Roger-Champion',
    'superdist(content, "Roger", "Nadal", para, ,)
    AS Tennis-Rivalry',
    'contain(content, regex"[A|a]shes", 2,)
    AS Aus-Eng-Cricket',
    'superdist(content, "Australia", "won", nw5, ,)
    AS Aus-victory'
  )
  Accumulate ('id')
) AS dt ORDER BY id;
```

### Output

```
id tag
-- -----
1 Chennai-Flood-Disaster
2 Roger-Champion
3 Tennis-Rivalry
4
5 Aus-Eng-Cricket,Aus-victory
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextTagger Example: Text, Unicode Emoticons (Emojis)

You can run queries with emojis only from the bteq prompt, not using Teradata Studio™.

### Input

```
=====
Input
=====
```

id	title_1	
1	Chennai Floods	Chennai floods have battered the capital city of Tamil Nadu and its adjoining
2	Tennis Superstars	Roger Federer born on 8 August 1981, is a greatest 🏆 tennis player, who has
3	Sports Rivalry	The Federer Nadal rivalry, known by many as Fedal, is between two profession
4	Sports Rivalry	The India Pakistan cricket rivalry is one of the most intense 😞 sports riva
5	Sports Rivalry	An Ashes series is traditionally of five Tests, hosted in turn by England an

(5 rows)

## SQL Call

```
SELECT * FROM TextTagger(
ON text_inputs_emojis
USING
TaggingRules (
'contain(contents, "👍", 1,) AS Thumbs',
'contain(contents, "🏆", 1,) or
  contain(contents, "greatest", 1,) AS Fabulous',
'contain(contents, "greatest", 1,) AS Wonderful',
'contain(contents, "😞", 1,) AS Weary',
'contain(title_1, "Tennis", 1,) and
  contain(contents, "Roger", 1,) AS Tennis-Greats',
'contain(contents, "India", 1,) and
  contain(contents, "Pakistan", 1,) AS Cricket-Rivalry',
'contain(contents, "Australia", 1,) and
  contain(contents, "England", 1,) AS The-Ashes'
)
OutputByTag ('true')
Accumulate ('id')
) AS dt ORDER BY id;
```

## Output

id	tag
1	Weary
2	Wonderful
2	Fabulous
2	Tennis-Greats
2	Thumbs
3	Wonderful
3	Fabulous
4	Cricket-Rivalry
5	The-Ashes

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TFIDF (ML Engine)

TF-IDF stands for "term frequency-inverse document frequency," a technique for evaluating the importance of a specific term in a specific document in a document set. Term frequency (tf) is the number of times that the term appears in the document and inverse document frequency (idf) is the number of times that the term appears in the document set. The TF-IDF score for a term is  $tf * idf$ . A term with a high TF-IDF score is especially relevant to the specific document.

The TFIDF function can do either of the following:

- Take any document set and output the inverse document frequency (IDF) and term frequency-inverse document frequency (TF-IDF) scores for each term.
- Use the output of a previous run of the TFIDF function on a training document set to predict TFIDF scores of an input (test) document set.

You can use the TF-IDF scores as input for many document clustering and classification algorithms, including:

- Cosine-similarity
- Latent Dirichlet allocation
- K-means clustering
- K-nearest neighbors

You can use the TF-IDF scores derived from a training document set to create a model in a classification function (for example, SVMSParse (ML Engine)) and then use the resulting TF-IDF scores in a classification prediction function (for example, SVMSParsePredict\_MLE (ML Engine)).

The TFIDF function represents each document as an N-dimensional vector, where N is the number of terms in the document set (therefore, the document vector is usually very sparse). Each entry in the document vector is the TF-IDF score of a term.

## TFIDF Syntax

### TFIDF version 2.3, TF version 1.2

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  [ ON (SELECT COUNT (DISTINCT docid) FROM doccount_table) AS DocCount DIMENSION ]
  [ ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
  ]
  [ ON (SELECT DISTINCT (term) AS term, idf FROM tf_idf_output_table)
    AS IDF PARTITION BY term
  ]
) AS alias;
```

## Large Document Sets

For large documents sets, the DocPerTerm table is required.

For training, this is the syntax for large document sets:

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  ON (SELECT COUNT (DISTINCT docid) FROM doccount_table) AS DocCount DIMENSION
  ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
) AS alias ORDER BY docid;
```

For prediction, this is the syntax for large document sets:

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  [ ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
  ]
  [ ON (SELECT DISTINCT (term) AS term, idf FROM tf_idf_output_table)
    AS IDF PARTITION BY term
  ]
) AS alias ORDER BY docid;
```

## Small Document Sets

This syntax is acceptable for small document sets:

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
  ) AS TF PARTITION BY term
  ON (SELECT COUNT (DISTINCT docid) FROM input_table) AS DocCount DIMENSION
) AS alias ORDER BY docid;
```

## TFIDF Syntax Elements

Formula

[Optional] Specify the formula for calculating the term frequency (tf) of term t in document d:

Option	Description
'normal' (Default)	<p>Normalized frequency:</p> $tf(t,d) = f(t,d) / \sum \{w, w \in d\}$ <p>This value is rf divided by number of terms in document.</p>
'bool'	<p>Boolean frequency:</p> $tf(t,d) = 1 \text{ if } t \text{ occurs in } d; \text{ otherwise, } tf(t,d) = 0.$
'log'	<p>Logarithmically-scaled frequency:</p> $tf(t,d) = \log(f(t,d)+1)$ <p>where <math>f(t,d)</math> is the number of times <math>t</math> occurs in <math>d</math> (that is, raw frequency, rf).</p>
'augment'	<p>Augmented frequency, which prevents bias towards longer documents:</p> $tf(t,d) = 0.5 + (0.5 \times f(t,d) / \max \{f(w,d) : w \in d\})$ <p>This value is rf divided by maximum raw frequency of any term in document.</p>

When using the output of a previous run of the TFIDF function on a training document set to predict TFIDF scores on an input document set, use the same Formula value for the input document set that you used for the training document set.

## TFIDF Input

The TFIDF function always requires as input the output of the TF function. Whether the other TFIDF input tables are required or optional depend on your reason for running the function.

Table	Description
TF	TF function input; document set.
DocCount	Required if running function to output IDF and TF-IDF values for each term in document set.
DocPerTerm	<p>Optional if running function to output IDF and TF-IDF values for each term in document set.</p> <p>If you omit this table, the function creates it by processing the entire document set, which can require a large amount of memory. If there is not enough memory to process the entire document set, the DocPerTerm table is required.</p>

Table	Description
IDF	<p>Required if running function to predict TF-IDF scores.</p> <p>This table is the output of an earlier call to TFIDF, using the training document set as input to the TF function, the DocCount table, and optionally, the DocPerTerm table.</p>

## TF Schema

Column	Data Type	Description
docid	Any	Document identifier.
term	VARCHAR	Term.
count	INTEGER	Number of times that term appears in document.

## TF Output and TFIDF Input Table Schema

Column	Data Type	Description
docid	Any	Document identifier.
term	VARCHAR	Term.
tf	DOUBLE PRECISION	Term frequency.
count	INTEGER	Number of times that term appears in document.

## DocCount Schema

Column	Data Type	Description
count	BIGINT	Number of documents in document set.

## DocPerTerm Schema

Column	Data Type	Description
term	VARCHAR	Term.
count	BIGINT	Number of documents that contain term.

## TFIDF Output

### Output Schema

Column	Data Type	Description
docid	Any	Document identifier of document d.
term	VARCHAR	Term t.
tf	DOUBLE PRECISION	Term frequency of term t in document d, calculated as specified by Formula syntax element.
idf	DOUBLE PRECISION	Inverse document frequency of term t in document d, calculated by this formula:  $IDF(t) = \log (\text{doccount} / \text{doccount}(t))$ where doccount is the number of documents in the document set and doccount (t) is the number of documents that contain the term t.
tf_idf	DOUBLE PRECISION	TFIDF score of term t in document d, calculated by this formula:  $TFIDF(t, d) = TF(t, d) * IDF(t)$

### TFIDF Examples

#### TFIDF Example: Tokenized Training Document Set

This example uses the NGramSplitter\_MLE function to tokenize a training document set, from which it creates the input table for the TFIDF function.

#### NGramSplitter\_MLE Input: tfidf\_train

docid	content
1	Chennai floods have battered the capital city of Tamil Nadu and its adjoining areas. Normal life came to a standstill when roads were submerged in water and all modes of transport were severely affected. In the past, Chennai has had tsunamis and earthquakes
2	Roger Federer born on 8 August 1981, is a greatest tennis player, who has been continuously ranked inside the top 10 since October 2002 and has won Wimbledon, USOpen, Australian and FrenchOpen titles mutiple times

docid	content
3	The Federer Nadal rivalry, known by many as Fedal, is between two professional tennis players, Roger Federer of Switzerland and Rafael Nadal of Spain. They are currently engaged in a storied rivalry, which many consider to be the greatest in tennis history. They have played 34 times, most recently in the 2015 Swiss Indoors final, and Nadal leads their eleven-year-old rivalry with an overall record of 23-11
4	The India Pakistan cricket rivalry is one of the most intense sports rivalries in the world. An India-Pakistan cricket match has been estimated to attract up to one billion viewers, according to TV ratings firms and various other reports. The 2011 World Cup semifinal between the two teams attracted around 988 million television viewers
5	An Ashes series is traditionally of five Tests, hosted in turn by England and Australia at least once every four years. As of August 2015, England hold the ashes, having won three of the five Tests in the 2015 Ashes series. Overall, Australia has won 32 series, England 32 and five series have been drawn.

## NGramSplitter\_MLE SQL Call

This call creates a table of tokenized input, tfidf\_token1, from tfidf\_train.

```
CREATE MULTiset TABLE tfidf_token1 AS (
  SELECT * FROM NGramSplitter_MLE (
    ON tfidf_train
    USING
    TextColumn ('content')
    Delimiter (' ')
    Grams ('1')
    Overlapping ('false')
    ConvertToLowerCase ('true')
    Punctuation ('\[.,?!\]\')
    Reset ('\[.,?!\]\')
    OutputTotalGramCount ('false')
    Accumulate ('docid')
  ) AS dt
) WITH DATA;
```

## SQL Call to Create TFIDF Input Table tfidf\_input1

The TFIDF input table must have the tokenized input in the column term.

```
CREATE MULTiset TABLE tfidf_input1 AS (
  SELECT docid, ngram AS term, frequency AS "count"
  FROM tfidf_token1 AS dt
) WITH DATA;
```

This query returns the following table:

```
SELECT * FROM tfidf_input1 ORDER BY 1, 3, 2;
```

docid	term	count
1	a	1
1	adjoining	1
1	affected	1
1	all	1
1	areas	1
1	battered	1
1	came	1
1	capital	1
1	city	1
1	earthquakes	1
1	floods	1
1	had	1
1	has	1
1	have	1
1	its	1
1	life	1
1	modes	1
1	nadu	1
1	normal	1
1	past	1
1	roads	1
1	severely	1
1	standstill	1
1	submerged	1
1	tamil	1
1	to	1
1	transport	1
1	tsunamis	1
1	water	1
1	when	1
1	chennai	2
1	in	2
1	of	2
1	the	2

1 were	2
1 and	3
2 10	1
2 1981	1
2 2002	1
2 8	1
2 a	1
2 august	1
2 australian	1
2 been	1
2 born	1
2 continuously	1
2 federer	1
2 frenchopen	1
2 greatest	1
2 inside	1
2 is	1
2 mutiple	1
2 october	1
2 on	1
2 player	1
2 ranked	1
2 roger	1
2 since	1
2 tennis	1
2 the	1
2 times	1
2 titles	1
2 top	1
2 usopen	1
2 who	1
2 wimbledon	1
2 won	1
2 and	2
2 has	2
3 2015	1
3 23â??11	1
3 34	1
3 a	1
3 an	1
3 are	1
3 as	1
3 be	1
3 between	1
3 by	1
3 consider	1
3 currently	1
3 eleven-year-old	1

3 engaged	1
3 fedal	1
3 final	1
3 greatest	1
3 have	1
3 history	1
3 indoors	1
3 is	1
3 known	1
3 leads	1
3 most	1
3 overall	1
3 played	1
3 players	1
3 professional	1
3 rafael	1
3 recently	1
3 record	1
3 roger	1
3 spain	1
3 storied	1
3 swiss	1
3 switzerland	1
3 their	1
3 times	1
3 to	1
3 two	1
3 which	1
3 with	1
3 and	2
3 federer	2
3 many	2
3 tennis	2
3 they	2
3 in	3
3 nadal	3
3 of	3
3 rivalry	3
3 the	3
4 2011	1
4 988	1
4 according	1
4 an	1
4 and	1
4 around	1
4 attract	1
4 attracted	1
4 been	1

4 between	1
4 billion	1
4 cup	1
4 estimated	1
4 firms	1
4 has	1
4 in	1
4 india	1
4 india-pakistan	1
4 intense	1
4 is	1
4 match	1
4 million	1
4 most	1
4 of	1
4 other	1
4 pakistan	1
4 ratings	1
4 reports	1
4 rivalries	1
4 rivalry	1
4 semifinal	1
4 sports	1
4 teams	1
4 television	1
4 tv	1
4 two	1
4 up	1
4 various	1
4 cricket	2
4 one	2
4 viewers	2
4 world	2
4 to	3
4 the	5
5 an	1
5 as	1
5 at	1
5 august	1
5 been	1
5 by	1
5 drawn	1
5 every	1
5 four	1
5 has	1
5 have	1
5 having	1
5 hold	1

5 hosted	1
5 is	1
5 least	1
5 once	1
5 overall	1
5 three	1
5 traditionally	1
5 turn	1
5 years	1
5 2015	2
5 32	2
5 and	2
5 australia	2
5 in	2
5 tests	2
5 won	2
5 ashes	3
5 england	3
5 five	3
5 of	3
5 the	3
5 series	4

## SQL Call to Create TFIDF Input Table tf1

```
CREATE MULTISET TABLE tf1 AS (
  SELECT * FROM tf (
    ON tfidf_input1 PARTITION BY docid
  ) AS dt1
) WITH DATA;
```

## TFIDF SQL Call

```
CREATE MULTISET TABLE tfidf_output1 AS (
  SELECT * FROM TFIDF (
    ON tf1 AS TF PARTITION BY term
    ON (
      SELECT CAST(COUNT(DISTINCT(docid)) AS integer) AS "count"
      FROM tfidf_input1
    ) AS DocCount DIMENSION
  ) AS dt
) WITH DATA;
```

## TFIDF Output

This query returns the following table:

```
SELECT * FROM tfidf_output1 ORDER BY tfidf DESC;
```

docid	term	tf	idf	tf_idf
5	series	0.07272727272727272	1.6094379124341003	0.11705002999520729
5	ashes	0.05454545454545454	1.6094379124341003	0.08778752249640547
5	england	0.05454545454545454	1.6094379124341003	0.08778752249640547
5	five	0.05454545454545454	1.6094379124341003	0.08778752249640547
1	chennai	0.046511627906976744	1.6094379124341003	0.07485757732251629
1	were	0.046511627906976744	1.6094379124341003	0.07485757732251629
3	nadal	0.04477611940298507	1.6094379124341003	0.07206438413884031
4	one	0.037037037037037035	1.6094379124341003	0.05960881157163334
4	cricket	0.037037037037037035	1.6094379124341003	0.05960881157163334
4	viewers	0.037037037037037035	1.6094379124341003	0.05960881157163334
4	world	0.037037037037037035	1.6094379124341003	0.05960881157163334
5	tests	0.03636363636363636	1.6094379124341003	0.058525014997603646
5	australia	0.03636363636363636	1.6094379124341003	0.058525014997603646
5	32	0.03636363636363636	1.6094379124341003	0.058525014997603646
3	many	0.029850746268656716	1.6094379124341003	0.04804292275922687
3	they	0.029850746268656716	1.6094379124341003	0.04804292275922687
2	mutiple	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	10	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	inside	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	on	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	player	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	october	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	frenchopen	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	since	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	titles	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	2002	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	continuously	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	australian	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	usopen	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	wimbledon	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	1981	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	who	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	ranked	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	born	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	8	0.02857142857142857	1.6094379124341003	0.04598394035526001
2	top	0.02857142857142857	1.6094379124341003	0.04598394035526001
3	rivalry	0.04477611940298507	0.9162907318741551	0.04102794321824575
1	earthquakes	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	submerged	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	past	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	transport	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	capital	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	city	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	battered	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	roads	0.023255813953488372	1.6094379124341003	0.03742878866125814
1	areas	0.023255813953488372	1.6094379124341003	0.03742878866125814

1 tamil	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 standstill	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 nadu	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 life	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 severely	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 adjoining	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 all	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 had	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 came	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 modes	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 its	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 affected	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 tsunamis	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 when	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 floods	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 normal	0.023255813953488372	1.6094379124341003	0.03742878866125814
1 water	0.023255813953488372	1.6094379124341003	0.03742878866125814
5 2015	0.03636363636363636	0.9162907318741551	0.033319662977242
5 won	0.03636363636363636	0.9162907318741551	0.033319662977242
4 around	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 teams	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 india	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 television	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 tv	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 estimated	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 other	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 india-pakistan	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 various	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 cup	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 ratings	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 988	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 attracted	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 up	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 rivalries	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 reports	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 billion	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 attract	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 match	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 million	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 sports	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 pakistan	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 semifinal	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 intense	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 firms	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 2011	0.018518518518518517	1.6094379124341003	0.02980440578581667
4 according	0.018518518518518517	1.6094379124341003	0.02980440578581667
5 having	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 turn	0.01818181818181818	1.6094379124341003	0.029262507498801823

5 drawn	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 four	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 traditionally	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 years	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 hosted	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 at	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 hold	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 every	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 once	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 least	0.01818181818181818	1.6094379124341003	0.029262507498801823
5 three	0.01818181818181818	1.6094379124341003	0.029262507498801823
4 to	0.05555555555555555	0.5108256237659907	0.028379201320332816
3 tennis	0.029850746268656716	0.9162907318741551	0.027351962145497167
3 federer	0.029850746268656716	0.9162907318741551	0.027351962145497167
2 won	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 roger	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 federer	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 greatest	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 times	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 tennis	0.02857142857142857	0.9162907318741551	0.02617973519640443
2 august	0.02857142857142857	0.9162907318741551	0.02617973519640443
3 played	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 spain	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 swiss	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 leads	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 are	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 with	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 34	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 recently	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 history	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 indoors	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 storied	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 23â??11	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 engaged	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 switzerland	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 consider	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 currently	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 record	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 which	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 fedal	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 their	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 rafael	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 be	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 players	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 known	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 eleven-year-old	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 final	0.014925373134328358	1.6094379124341003	0.024021461379613435
3 professional	0.014925373134328358	1.6094379124341003	0.024021461379613435

4 between	0.018518518518518517	0.9162907318741551	0.016968346886558426
4 two	0.018518518518518517	0.9162907318741551	0.016968346886558426
4 rivalry	0.018518518518518517	0.9162907318741551	0.016968346886558426
4 most	0.018518518518518517	0.9162907318741551	0.016968346886558426
5 overall	0.01818181818181818	0.9162907318741551	0.016659831488621
5 by	0.01818181818181818	0.9162907318741551	0.016659831488621
5 as	0.01818181818181818	0.9162907318741551	0.016659831488621
5 august	0.01818181818181818	0.9162907318741551	0.016659831488621
2 been	0.02857142857142857	0.5108256237659907	0.014595017821885449
2 a	0.02857142857142857	0.5108256237659907	0.014595017821885449
3 greatest	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 roger	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 as	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 overall	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 most	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 2015	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 times	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 between	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 two	0.014925373134328358	0.9162907318741551	0.013675981072748583
3 by	0.014925373134328358	0.9162907318741551	0.013675981072748583
2 has	0.05714285714285714	0.22314355131420976	0.0127510600750977
5 of	0.05454545454545454	0.22314355131420976	0.012171466435320532
1 a	0.023255813953488372	0.5108256237659907	0.011879665668976528
1 have	0.023255813953488372	0.5108256237659907	0.011879665668976528
1 to	0.023255813953488372	0.5108256237659907	0.011879665668976528
1 in	0.046511627906976744	0.22314355131420976	0.010378769828567896
1 of	0.046511627906976744	0.22314355131420976	0.010378769828567896
3 in	0.04477611940298507	0.22314355131420976	0.009991502297651183
3 of	0.04477611940298507	0.22314355131420976	0.009991502297651183
4 been	0.018518518518518517	0.5108256237659907	0.009459733773444272
4 an	0.018518518518518517	0.5108256237659907	0.009459733773444272
5 been	0.01818181818181818	0.5108256237659907	0.009287738613927104
5 have	0.01818181818181818	0.5108256237659907	0.009287738613927104
5 an	0.01818181818181818	0.5108256237659907	0.009287738613927104
5 in	0.03636363636363636	0.22314355131420976	0.008114310956880354
3 have	0.014925373134328358	0.5108256237659907	0.007624263041283444
3 to	0.014925373134328358	0.5108256237659907	0.007624263041283444
3 an	0.014925373134328358	0.5108256237659907	0.007624263041283444
3 a	0.014925373134328358	0.5108256237659907	0.007624263041283444
2 is	0.02857142857142857	0.22314355131420976	0.00637553003754885
1 has	0.023255813953488372	0.22314355131420976	0.005189384914283948
4 in	0.018518518518518517	0.22314355131420976	0.004132287987300181
4 has	0.018518518518518517	0.22314355131420976	0.004132287987300181
4 is	0.018518518518518517	0.22314355131420976	0.004132287987300181
4 of	0.018518518518518517	0.22314355131420976	0.004132287987300181
5 has	0.01818181818181818	0.22314355131420976	0.004057155478440177
5 is	0.01818181818181818	0.22314355131420976	0.004057155478440177
3 is	0.014925373134328358	0.22314355131420976	0.0033305007658837277

4 the	0.09259259259259259	0.0	0.0
4 and	0.018518518518518517	0.0	0.0
2 the	0.02857142857142857	0.0	0.0
2 and	0.05714285714285714	0.0	0.0
3 and	0.029850746268656716	0.0	0.0
3 the	0.04477611940298507	0.0	0.0
5 the	0.05454545454545454	0.0	0.0
5 and	0.03636363636363636	0.0	0.0
1 the	0.046511627906976744	0.0	0.0
1 and	0.06976744186046512	0.0	0.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

### TFIDF Example: Tokenized Test Set

This example uses the IDF values from `tfidf_output1`, output by TFIDF Example: Tokenized Training Document Set to predict the TFIDF scores of a test document set.

### NGramSplitter\_MLE Input: `tfidf_test`

docid	content
6	In Chennai, India, floods have closed roads and factories, turned off power, shut down the airport and forced thousands of people out of their homes.
7	Spanish tennis star Rafael Nadal said he was happy with the improvement in his game after a below-par year, and looked forward to reigniting his long-time rivalry with Roger Federer in India.
8	Nadal, the world number five, said he has always enjoyed playing against Federer and hoped they would do so for years to come.

### NGramSplitter\_MLE SQL Call

This call creates a table of tokenized input, `tfidf_token1`, from `tfidf_test`.

```
CREATE MULTISET TABLE tfidf_token1 AS (
  SELECT * FROM NGramSplitter_MLE (
    ON tfidf_test
    USING
    TextColumn ('content')
    Delimiter (' ')
    Grams ('1')
    Overlapping ('false')
    ConvertToLowerCase ('true')
    Punctuation ('\[,.\,?\!\\\')
```

```

Reset ('\[.,?!\]')
OutputTotalGramCount ('false')
Accumulate ('docid')
) AS dt
) WITH DATA;

```

## SQL Call to Create TFIDF Input Table tfidf\_input1

```

CREATE MULTISET TABLE tfidf_input1 AS (
  SELECT docid, ngram AS term, frequency AS "count" FROM tfidf_token1 AS dt
) WITH DATA;

```

## SQL Call to Create TFIDF Input Table tf1

```

CREATE MULTISET TABLE tf1 AS (
  SELECT * FROM tf (
    ON tfidf_input1 PARTITION BY docid
    USING
    Formula ('normal')
  ) AS dt1
) WITH DATA;

```

## TFIDF SQL Call

```

CREATE MULTISET TABLE tfidf_output2 AS (
  SELECT * FROM TFIDF (
    ON tf2 AS TF PARTITION BY TERM
    ON (SELECT CAST(COUNT(DISTINCT(docid)) AS INTEGER) AS "count"
    FROM tfidf_output1) AS DocCount DIMENSION
  ) AS dt
) WITH DATA;

```

## TFIDF Output

This query returns the following table:

```
SELECT * FROM tfidf_output2 ORDER BY tf_idf DESC;
```

docid	term	tf	idf	tf_idf
6	of	0.08	1.6094379124341003	0.128755032994728
7	his	0.0625	1.6094379124341003	0.10058986952713127
7	with	0.0625	1.6094379124341003	0.10058986952713127
8	so	0.043478260869565216	1.6094379124341003	0.06997556141017827
8	world	0.043478260869565216	1.6094379124341003	0.06997556141017827
8	always	0.043478260869565216	1.6094379124341003	0.06997556141017827
8	enjoyed	0.043478260869565216	1.6094379124341003	0.06997556141017827
8	they	0.043478260869565216	1.6094379124341003	0.06997556141017827

8 come	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 has	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 playing	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 hoped	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 years	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 number	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 against	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 would	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 five	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 do	0.043478260869565216	1.6094379124341003	0.06997556141017827
8 for	0.043478260869565216	1.6094379124341003	0.06997556141017827
6 chennai	0.04	1.6094379124341003	0.064377516497364
6 down	0.04	1.6094379124341003	0.064377516497364
6 turned	0.04	1.6094379124341003	0.064377516497364
6 factories	0.04	1.6094379124341003	0.064377516497364
6 their	0.04	1.6094379124341003	0.064377516497364
6 people	0.04	1.6094379124341003	0.064377516497364
6 have	0.04	1.6094379124341003	0.064377516497364
6 off	0.04	1.6094379124341003	0.064377516497364
6 airport	0.04	1.6094379124341003	0.064377516497364
6 thousands	0.04	1.6094379124341003	0.064377516497364
6 forced	0.04	1.6094379124341003	0.064377516497364
6 out	0.04	1.6094379124341003	0.064377516497364
6 roads	0.04	1.6094379124341003	0.064377516497364
6 shut	0.04	1.6094379124341003	0.064377516497364
6 power	0.04	1.6094379124341003	0.064377516497364
6 closed	0.04	1.6094379124341003	0.064377516497364
6 floods	0.04	1.6094379124341003	0.064377516497364
6 homes	0.04	1.6094379124341003	0.064377516497364
7 in	0.0625	0.9162907318741551	0.057268170742134694
7 star	0.03125	1.6094379124341003	0.050294934763565634
7 after	0.03125	1.6094379124341003	0.050294934763565634
7 long-time	0.03125	1.6094379124341003	0.050294934763565634
7 improvement	0.03125	1.6094379124341003	0.050294934763565634
7 was	0.03125	1.6094379124341003	0.050294934763565634
7 looked	0.03125	1.6094379124341003	0.050294934763565634
7 reigniting	0.03125	1.6094379124341003	0.050294934763565634
7 rafael	0.03125	1.6094379124341003	0.050294934763565634
7 spanish	0.03125	1.6094379124341003	0.050294934763565634
7 forward	0.03125	1.6094379124341003	0.050294934763565634
7 year	0.03125	1.6094379124341003	0.050294934763565634
7 rivalry	0.03125	1.6094379124341003	0.050294934763565634
7 happy	0.03125	1.6094379124341003	0.050294934763565634
7 tennis	0.03125	1.6094379124341003	0.050294934763565634
7 a	0.03125	1.6094379124341003	0.050294934763565634
7 below-par	0.03125	1.6094379124341003	0.050294934763565634
7 game	0.03125	1.6094379124341003	0.050294934763565634
7 roger	0.03125	1.6094379124341003	0.050294934763565634

```

6 and                0.08 0.5108256237659907 0.04086604990127926
8 to                 0.043478260869565216 0.9162907318741551 0.039838727472789354
8 he                 0.043478260869565216 0.9162907318741551 0.039838727472789354
8 nadal             0.043478260869565216 0.9162907318741551 0.039838727472789354
8 said              0.043478260869565216 0.9162907318741551 0.039838727472789354
8 federer           0.043478260869565216 0.9162907318741551 0.039838727472789354
6 india             0.04 0.9162907318741551 0.03665162927496621
6 in                0.04 0.9162907318741551 0.03665162927496621
7 nadal             0.03125 0.9162907318741551 0.028634085371067347
7 to                0.03125 0.9162907318741551 0.028634085371067347
7 india             0.03125 0.9162907318741551 0.028634085371067347
7 he                0.03125 0.9162907318741551 0.028634085371067347
7 federer           0.03125 0.9162907318741551 0.028634085371067347
7 said              0.03125 0.9162907318741551 0.028634085371067347
8 the               0.043478260869565216 0.5108256237659907 0.022209809728956118
8 and               0.043478260869565216 0.5108256237659907 0.022209809728956118
6 the               0.04 0.5108256237659907 0.02043302495063963
7 and               0.03125 0.5108256237659907 0.01596330074268721
7 the               0.03125 0.5108256237659907 0.01596330074268721

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Named Entity Recognition (NER) Functions (ML Engine)

*Named entity recognition (NER)* is a process for finding specified entities in text. For example, a simple news named-entity recognizer for English might find the person "John J. Smith" and the location "Seattle" in the text string "John J. Smith lives in Seattle."

NER functions let you specify how to extract named entities when training the data models. ML Engine provides two sets of NER functions:

Function Set	Supported Languages
NER Functions (CRF Model Implementation)	English, simplified Chinese, traditional Chinese
NER Functions (Maximum Entropy Model Implementation)	English

## NER Functions (CRF Model Implementation)

Function	Description
NERTrainer (ML Engine)	Takes training data and outputs CRF model (binary file).

Function	Description
NERExtractor (ML Engine)	<p>Takes input documents and extracts specified entities, using one or more CRF models and, if appropriate, rules (regular expressions) or a dictionary.</p> <p>Uses models to extract names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and dictionary to extract known entities.</p>
NEREvaluator (ML Engine)	Evaluates CRF model.

The CRF model implementation supports English, simplified Chinese, and traditional Chinese text.

Related information

Related information

NER Functions (Maximum Entropy Model Implementation)

## NERTrainer (ML Engine)

The NERTrainer function takes training data and outputs a CRF model (a binary file) that can be specified in the function NERExtractor (ML Engine) and NEREvaluator (ML Engine).

NERTrainer uses files that are preinstalled on ML Engine. For details, see Preinstalled Files That Functions Use.

## NERTrainer Syntax

## Version 1.8

```
SELECT * FROM NERTrainer (
  ON { table | view | (query) } PARTITION BY 1
  USING
  ModelFileName (model_file)
  TextColumn ('text_column')
  [ ExtractorJAR ('jar_file') ]
  FeatureTemplate ('template_file')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ MaxIterNum (max_iteration_times) ]
  [ Eta (eta_threshold_value) ]
  [ MinOccurNum (threshold_value) ]
) AS alias;
```

## NERTrainer Syntax Elements

### ModelFileName

Specify the name of the model file that the function creates and installs on ML Engine.

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### ExtractorJAR

[Optional] Specify the name of the JAR file that contains the Java classes that extract features. You must install this JAR file on ML Engine before calling the function.

The name `jar_file` is case-sensitive.

ML Engine does not support the creation of new extractor classes. However, it does support existing JAR files—for installation instructions, see *Teradata Vantage™ User Guide*, B700-4002.

Default behavior: The function uses only the predefined extractor classes.

### FeatureTemplate

Specify the name of the file that specifies how to create features when training the model.

### InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

### MaxIterNum

[Optional] Specify the maximum number of iterations.

Default: 1000

### Eta

[Optional] Specify the tolerance of the termination criterion. Defines the differences of the values of the loss function between two sequential epochs.

When training a model, the function performs  $n$ -times iterations. At the end of each epoch, the function calculates the loss or cost function on the training samples. If the loss function value change is very small between two sequential epochs, the function considers the training process to have converged.

The function defines Eta as:

$$\text{Eta} = (f(n) - f(n-1)) / f(n-1)$$

where  $f(n)$  is the loss function value of the  $n$ th epoch.

Default: 0.0001

### MinOccurNum

[Optional] Specify the minimum number times that a feature must occur in the input text before the function uses the feature to construct the model.

Default: 0

## NERTrainer Input

### Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Text to analyze. Within text, each entity must be identified with this syntax: <START:entity_type>entity<END> For example: <START:location>Country1<END> has arrived

## NERTrainer Output

The function outputs a message and a CRF model (a binary file installed on ML Engine).

### Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Reports training time and file size of model.

## NERTrainer Example

### Input

- Input table: ner\_sports\_train, a collection of sports news items (500 rows)
- Feature template file: template\_1.txt, which is preinstalled on ML Engine.

ner\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOC> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top

id	content
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOC> Grace Road <END>
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
...	...

## SQL Call

```
SELECT * FROM NERTrainer (
  ON ner_sports_train PARTITION BY 1
  USING
    TextColumn ('content')
    FeatureTemplate ('template_1.txt')
    OutputModelFile ('ner_model.bin')
) AS dt;
```

## Output

```
train_result
-----
Model generated.
Training time(s): 3.129
File size(KB): 374
Model successfully installed.
```

The model file, ner\_model.bin, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NERExtractor (ML Engine)

The NERExtractor function takes input documents and extracts specified entities, using one or more CRF models (output by the function NERTrainer (ML Engine)) and, if appropriate, rules (regular expressions) or a dictionary.

The function uses models to extract the names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and a dictionary to extract known entities.

NERExtractor uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NERExtractor Syntax

### Version 1.8

```
SELECT * FROM NERExtractor (
  ON input_table PARTITION BY { ANY | key }
  [ ON rules_table AS Rules DIMENSION ]
  [ ON dictionary_table AS Dict DIMENSION ]
  USING
  TextColumn ('text_column')
  [ InputModelFiles ('input_model_file[:jar_file]' [,...]) ]
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ ShowContext ('n') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related information

Related information  
Column Specification Syntax Elements

### NERExtractor Syntax Elements

TextColumn

Specify the name of the input table column that contains the text to analyze.

InputModelFiles

[Optional] Specify the CRF models (binary files) to use, output by NERTrainer (ML Engine). If you specified the ExtractorJAR syntax element in the NERTrainer call that created input\_model\_file, then you must specify the same jar\_file in this syntax element. You must install input\_model\_file and jar\_file in ML Engine before calling the NERExtractor function.

The names input\_model\_file and jar\_file are case-sensitive.

InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

ShowContext

[Optional] Specify the number of context words to output (a positive integer). The function outputs the n words that precede the entity, the entity, and the n words that follow the entity.

Default: 0

Accumulate

[Optional] Specify the names of the input table columns to copy to the output table.

## NERExtractor Input

Table	Description
Input table	Text to analyze. Tip: To optimize function performance, remove punctuation marks from text with TextParser (ML Engine) function.
Rules	[Optional] Rules to use when extracting entities from text.
Dict	[Optional] Dictionary to use when extracting entities from text.

## Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Text to analyze.
accumulate_column	Any	Column to copy to output table.

## Rules Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
regex	VARCHAR	Regular expression that represents an entity of this type. Expression must conform to Java Regex standard, documented at <a href="http://docs.oracle.com/javase/tutorial/essential/regex/quant.html">http://docs.oracle.com/javase/tutorial/essential/regex/quant.html</a> .

## Dict Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
dict	VARCHAR	Dictionary word.

**NERExtractor Output****Output Table Schema**

Column	Data Type	Description
accumulate_column	Same as in input table	Column copied from input table.
sn	INTEGER	Serial number of extracted entity.
entity	VARCHAR	Extracted entity.
type	VARCHAR	Type of extracted entity.
start	INTEGER	Start position of extracted entity in input text.
end	INTEGER	End position of extracted entity in input text.
context	VARCHAR	[Column appears only with ShowContent syntax element.] Context of extracted entity.
approach	VARCHAR	Method used to identify extracted entity—CRF, RULE, or DICT.

**NERExtractor Example****Input**

- Input table: ner\_sports\_test2, which contains text to analyze.
- Rules: rule\_table, which is preinstalled on ML Engine.
- Model: ner\_model.bin, output by NERTrainer Example.

Input table: ner\_sports\_test2

id	content
528	email sports@espn.com to contact for all sport info
529	email cricket@espn.com to contact for all cricket info
530	email tennis@espn.com to contact for all tennis info
531	1= Igor Trandenkov (Russia) 5.86
532	3. Maksim Tarasov (Russia) 5.86
533	4. Tim Lobinger (Germany) 5.80
534	5. Igor Potapovich (Kazakstan) 5.80
535	6. Jean Galfione (France) 5.65
536	7. Pyotr Bochkary (Russia) 5.65
537	8. Dmitri Markov (Belarus) 5.65
583	GENEVA 1996-08-30
584	UEFA came down heavily on Belgian club Standard Liege on Friday for disgraceful behaviour in an Intertoto final match against Karlsruhe of Germany .
585	The Belgian club were fined 25
586	He was sent off for insulting the referee and then urged his team mates to protest .

id	content
587	Roberto Bisconti will be sidelined for six Euro ties after pushing the referee in the back as he protested about a Karlsruhe goal
588	Karlsruhe won the August 20 match 3-1 thanks to two late goals .
589	They took the tie 3-2 on aggregate and qualified for the UEFA Cup .
591	ATHLETICS - HARRISON
592	MONTE CARLO 1996-08-30
593	Olympic champion Kenny Harrison and world record holder Jonathan Edwards will both take part in a triple jump competition at the Solidarity Meeting for Sarajevo on September 9 .
594	The International Amateur Athletic Federation said on Friday that a schedule reshuffle had allowed organisers to hold a men s triple jump as well as the women s long jump on the one usable runway at the war-devastated Kosevo stadium .
595	Atlanta Games silver medal winner Edwards has called on other leading athletes to take part in the Sarajevo meeting -- a goodwill gesture towards Bosnia as it recovers from the war in the Balkans -- two days after the grand prix final in Milan .
596	Edwards was quoted as saying : What type of character do we show by going to the IAAF Grand Prix Final in Milan where there is a lot of money to make but refusing to make the trip to Sarajevo as a humanitarian gesture ?
598	SOCCER - BARATELLI TO COACH NICE .
599	NICE
600	Former international goalkeeper Dominique Baratelli is to coach struggling French first division side Nice
601	Baratelli
602	Nice have been unable to win any of their four league matches played this season and are lying a lowly 18th in the table .

Rules: rule\_table

type	regex
email	[\w\-\_]([\w\w])+[\w]+@([\w\-\_]+\.)+[a-zA-Z]{2,4}

## SQL Call

```

SELECT * FROM NERExtractor (
  ON ner_sports_test2 PARTITION BY ANY
  ON rule_table AS Rules DIMENSION
  USING
    TextColumn ('content')
    InputModelFiles ('ner_model.bin')
    ShowContext (2)
    Accumulate ('id')
) AS dt ORDER BY id, sn;

```

## Output

id	sn	entity	type_ner	start_ner	end_ner	context
528	1	sports@espn.com	email	2	2	... email sports@espn.com to contact
529	1	cricket@espn.com	email	2	2	... email cricket@espn.com to contact
530	1	tennis@espn.com	email	2	2	... email tennis@espn.com to contact
531	1	Igor Trandekov	PER	2	3	... 1= Igor Trandekov (Russia) 5.86
532	1	Maksim Tarasov	PER	2	3	... 3. Maksim Tarasov (Russia) 5.86
533	1	Tim Lobinger	PER	2	3	... 4. Tim Lobinger (Germany) 5.80
534	1	Igor Potapovich	PER	2	3	... 5. Igor Potapovich (Kazakstan) 5.80
535	1	Jean Galfione	PER	2	3	... 6. Jean Galfione (France) 5.65
536	1	Pyotr Bochkary	PER	2	3	... 7. Pyotr Bochkary (Russia) 5.65
537	1	Dmitri Markov	PER	2	3	... 8. Dmitri Markov (Belarus) 5.65
583	1	GENEVA	LOC	1	1	... ... GENEVA 1996-08-30 ...
584	1	Standard Liege	PER	8	9	Belgian club Standard Liege on Friday
587	1	Roberto Bisconti	PER	1	2	... ... Roberto Bisconti will be
591	1	HARRISON	PER	3	3	ATHLETICS - HARRISON ... ...
592	1	MONTE CARLO	PER	1	2	... ... MONTE CARLO 1996-08-30 ...
593	1	Kenny Harrison	PER	3	4	Olympic champion Kenny Harrison and world
593	2	Jonathan Edwards	PER	9	10	record holder Jonathan Edwards will both
596	1	What	ORG	7	7	saying : What type of
598	1	BARATELLI TO	PER	3	4	SOCCER - BARATELLI TO COACH NICE
599	1	NICE	PER	1	1	... ... NICE ... ...
600	1	Dominique Baratelli	PER	4	5	international goalkeeper Dominique Baratelli is
600	2	Nice	PER	14	14	division side Nice ... ...
601	1	Baratelli	PER	1	1	... ... Baratelli ... ...

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NEREvaluator (ML Engine)

The NREvaluator function evaluates a CRF model (output by the function NERTrainer (ML Engine)).

NEREvaluator uses files that are preinstalled on ML Engine. For details, see Preinstalled Files That Functions Use.

### NEREvaluator Syntax

## Version 1.9

```
SELECT * FROM NREvaluator (
  ON { table | view | (query) } PARTITION BY 1
  USING
  TextColumn ('text_column')
  ModelFile ('model_file[:jar_file]')
```

```
[ InputLanguage ( { 'en' | 'zh_CN' | 'zh_TW' } ) ]
) AS alias;
```

## NEREvaluator Syntax Elements

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### ModelFile

Specify the CRF model file to evaluate, created and automatically installed by NERTrainer (ML Engine).

If you specified the ExtractorJAR syntax element in the NERTrainer call that created model\_file, then you must specify the same jar\_file in this syntax element. You must install the jar\_file on ML Engine before calling the NERExtractor function.

The names model\_file and jar\_file are case-sensitive.

### InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

## NEREvaluator Input

The input table has the same schema as the NERExtractor Input table.

## NEREvaluator Output

### Output Table Schema

Column	Data Type	Description
type	VARCHAR	Entity type. Final row value: -AVG-
precision	DOUBLE PRECISION	Precision value of the entity type. Final row value: Average precision value for all entity types.
recall	DOUBLE PRECISION	Recall value of the entity type. Final row value: Average recall value for all entity types.

Column	Data Type	Description
f1_measure	DOUBLE PRECISION	F1 score (F-measure) of the entity type. Final row value: Average F1 score for all entity types.

### NEREvaluator Example

This function evaluates the efficacy of the model file ner\_model.bin, created by the NERTrainer function in terms of precision, recall, and f1\_measure.

### Input

- ner\_model.bin, output by NERTrainer Example

### SQL Call

```
SELECT * FROM NEREvaluator (
  ON ner_sports_test2 PARTITION BY 1
  USING
    TextColumn ('content')
    ModelFile ('ner_model.bin')
) AS dt;
```

### Output

```
type_ner precision_ner recall f1_measure
-----
LOC          1 0.4444    0.6154
ORG          0      0     -1
PER         0.7222 0.8125    0.7647
-AVG-        0.7778 0.4884     0.6
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NER Functions (Maximum Entropy Model Implementation)

Function	Description
NamedEntityFinderTrainer (ML Engine)	Takes training data and outputs a maximum entropy model (binary file).

Function	Description
NamedEntityFinder (ML Engine)	<p>Evaluates input, identifies tokens based on specified model, and outputs tokens with detailed information.</p> <p>Uses model to extract entity types 'PERSON', 'LOCATION', and 'ORGANIZATION' and rules to extract entity types 'DATE', 'TIME', 'EMAIL' and 'MONEY'. If you specify these entity names, the function invokes the default model types and model file names. To extract all entities in one NamedEntityFinder call, specify 'ALL'.</p>
Named Entity Finder Evaluator (ML Engine)	Evaluates maximum entropy model.

The maximum entropy model implementation supports only English text.

Related information

Related information

NER Functions (CRF Model Implementation)

## NamedEntityFinderTrainer (ML Engine)

The NamedEntityFinderTrainer function takes training data and outputs a Maximum Entropy data model. The function is based on OpenNLP, and follows its annotation. For more information on OpenNLP, see <https://opennlp.apache.org/docs/1.8.4/manual/opennlp.html>.

The trainer supports only the English language.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see Preinstalled Files That Functions Use.

## NamedEntityFinderTrainer Syntax

### Version 1.7

```
SELECT * FROM NamedEntityFinderTrainer (
  ON { table | view | (query) } PARTITION BY 1 [ ORDER BY order_column ]
  USING
  OutputModelFile (output_model_file)
  TextColumn ('text_column')
  EntityType ('entity_type')
  [ IterNum (iterator)]
  [ Cutoff (cutoff)]
) AS alias;
```

For repeatable results, you must specify ORDER BY and order\_column must have a unique value for each row.

## NamedEntityFinderTrainer Syntax Elements

### OutputModelFile

Specify the name of the data model file to create.

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### EntityType

Specify the entity type to train (for example, PERSON). The input training documents must contain the same tag.

### IterNum

[Optional] Specify the iterator number for training (an openNLP training parameter).

Default: 100

### Cutoff

[Optional] Specify the cutoff number for training (an openNLP training parameter).

Default: 5

## NamedEntityFinderTrainer Input

### Input Table Schema

Column	Data Type	Description
text_column	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax: <START:entity_type>entity<END> For example: <START:location>Country1<END> has arrived

## NamedEntityFinderTrainer Output

The function outputs a message and a Max Entropy model (a binary file automatically installed on ML Engine).

### Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Message indicating whether the function ran successfully.

## NamedEntityFinderTrainer Example

### Input

- Input Table: nermem\_sports\_train, which has 50 rows of sports news

Input Table: nermem\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderTrainer (
  ON nermem_sports_train PARTITION BY 1
  USING
    EntityType ('LOCATION')
    TextColumn ('content')
    OutputModelFile (location.sports)
) AS dt;
```

## Output

```
train_result
-----
model installed
```

The model table, location.sports, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NamedEntityFinder (ML Engine)

The NamedEntityFinder function evaluates the input, identifies tokens based on the specified model, and outputs the tokens with detailed information. The function does not identify sentences; it simply tokenizes. Token identification is not case-sensitive.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

### NamedEntityFinder Syntax

## Version 1.6

```
SELECT * FROM NamedEntityFinder (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON (configure_table) AS ConfigurationTable DIMENSION ]
  USING
  TextColumn ('text_column')
  [ Models ('entity_type[:model_type:{model_file|regular_expression}'][,...] | 'all' ) ]
  [ ShowContext ('context_words') ]
  [ EntityColName ('entity_column') ]
  [ Accumulate ( { 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related information

[Column Specification Syntax Elements](#)  
[Regular Expressions in Syntax Elements](#)

### NamedEntityFinder Syntax Elements

**TextColumn**

Specify the name of the input table column that contains the text to analyze.

**Models**

[Optional] Required if you do not specify ConfigurationTable, in which case you cannot specify 'all'. Specify the model items to load.

If you specify both ConfigurationTable and this syntax element, the function loads the specified model items from ConfigurationTable.

The entity\_type is the name of an entity type (for example, PERSON, LOCATION, or EMAIL), which appears in the output table.

model_type	Description
'max entropy'	Maximum entropy language model output by training.

model_type	Description
'rule'	Rule-based model, a plain text file with one regular expression on each line.
'dictionary'	Dictionary-based model, a plain text file with one word on each line.
'reg exp'	Regular expression that describes entity_type.

If model\_type is 'reg exp', specify regular\_expression (a regular expression that describes entity\_type); otherwise, specify model\_file (the name of the model file).

If you specify ConfigurationTable, you can use entity\_type as a shortcut. For example, if the ConfigurationTable has the row 'organization, max entropy, en-ner-organization.bin', you can specify Models ('organization') as a shortcut for Models ('organization:max entropy:en-ner-organization.bin').

For model\_type 'max entropy', if you specify ConfigurationTable and omit this syntax element, then the JVM of the worker node needs more than 2GB of memory.

Default: 'all' (If you specify ConfigurationTable but omit this syntax element.)

#### ShowContext

[Optional] Specify the number of context words to output. If context\_words is n (which must be a positive integer), the function outputs the n words that precede the entity, the entity, and the n words that follow the entity.

Default: 0

#### EntityColName

[Optional] Specify the name of the output table column that contains the entity names.

Default: 'entity'

#### Accumulate

[Optional] Specify the names of input columns to copy to the output table. No accumulate\_column can be an entity\_column.

Default: All input columns

## Creating the Table of Default Models

Before calling the NamedEntityFinder function, you must create the table of default models. To create the table, use this command:

```
DROP TABLE nameFind_configure;
```

```
CREATE MULTISET TABLE nameFind_configure (
  model_name VARCHAR(50),
  model_type VARCHAR(50),
  model_file VARCHAR(50)
);
```

Default English-language models are provided with the SQL functions. Before using these models, you must create a default configure\_table, as follows:

```
INSERT INTO nameFind_configure VALUES ('person','max entropy','en-ner-person.bin');
INSERT INTO nameFind_configure VALUES ('location','max entropy','en-ner-location.bin');
INSERT INTO nameFind_configure VALUES ('organization','max entropy','en-ner-organization.bin');
```

```

INSERT INTO nameFind_configure VALUES ('date','rules','date.rules');
INSERT INTO nameFind_configure VALUES ('time','rules','time.rules');
INSERT INTO nameFind_configure VALUES ('phone','rules','phone.rules');
INSERT INTO nameFind_configure VALUES ('money','rules','money.rules');
INSERT INTO nameFind_configure VALUES ('email','rules','email.rules');
INSERT INTO nameFind_configure VALUES ('percentage','rules','percentage.rules');

```

Default English-Language Models in Table

nameFind\_configure

model_name	model_type	model_file
person	max entropy	en-ner-person.bin
location	max entropy	en-ner-location.bin
organization	max entropy	en-ner-organization.bin
date	rules	date.rules
time	rules	time.rules
phone	rules	phone.rules
money	rules	money.rules
email	rules	email.rules
percentage	rules	percentage.rules

### NamedEntityFinder Input

## Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Contains input text.
accumulate_column	Any	Column to copy to output table.

## ConfigurationTable Schema

This table is optional.

Column	Data Type	Description
model_name	VARCHAR	Name of an entity type (for example, PERSON, LOCATION, or EMAIL).

Column	Data Type	Description										
model_type	VARCHAR	One of these model types:										
		<table><tr><th>model_type</th><th>Description</th></tr><tr><td>'max entropy'</td><td>Maximum entropy language model created by training</td></tr><tr><td>'rule'</td><td>Rule-based model, a plain text file with one regular expression on each line</td></tr><tr><td>'dictionary'</td><td>Dictionary-based model, a plain text file with one word on each line</td></tr><tr><td>'reg exp'</td><td>Regular expression that describes entity_type</td></tr></table>	model_type	Description	'max entropy'	Maximum entropy language model created by training	'rule'	Rule-based model, a plain text file with one regular expression on each line	'dictionary'	Dictionary-based model, a plain text file with one word on each line	'reg exp'	Regular expression that describes entity_type
		model_type	Description									
		'max entropy'	Maximum entropy language model created by training									
		'rule'	Rule-based model, a plain text file with one regular expression on each line									
'dictionary'	Dictionary-based model, a plain text file with one word on each line											
'reg exp'	Regular expression that describes entity_type											
model_file	VARCHAR	Name of model file that describes the entity type. This column appears if model_type is not 'reg exp'.										
reg_exp	VARCHAR	Regular expression that describes the entity type. This column appears if model_type is 'reg exp'.										

### NamedEntityFinder Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as in input table	Column copied from input table.
entity_type	VARCHAR	Entity type.
entity	VARCHAR	Entity name.
entity_start	INTEGER	[Column appears only with ShowEntityContext syntax element.] Start position.
entity_end	INTEGER	[Column appears only with ShowEntityContext syntax element.] End position.
context	VARCHAR	[Column appears only with ShowEntityContext syntax element.] Words before and after the entity.

### NamedEntityFinder Example

### Input

Input Table: assortedtext\_input

id	source	content
1001	misc	contact Alan by email at sports@espn.com for all sport info

id	source	content
1002	misc	contact Mark at cricket@espn.com for all cricket info
1003	misc	contact Roger at tennis@espn.com for all tennis info
1004	wiki	The contiguous United States consists of the 48 adjoining U.S. states plus Washington, D.C., on the continent of North America
1005	wiki	California's economy is centered on Technology, Finance, real estate services, Government, and professional, Scientific and Technical business Services; together comprising 58% of the State Government economy
1006	wiki	Houston is the largest city in Texas and the fourth-largest in the United States, while San Antonio is the second largest and seventh largest in the state.
1007	wiki	Thomas is a photographer whose natural landscapes of the West are also a statement about the importance of the preservation of the wildness

## SQL Call

```
SELECT * FROM NamedEntityFinder (
  ON assortedtext_input PARTITION BY ANY
  ON namefind_configure AS ConfigurationTable DIMENSION
  USING
    TextColumn ('content')
    Models ('all')
    Accumulate ('id', 'source')
) AS dt ORDER BY id;
```

## Output

id	source	entity	entity_type
1001	misc	sports@espn.com	email
1002	misc	cricket@espn.com	email

1002	misc	Mark	person
1003	misc	Roger	person
1003	misc	tennis@espn.com	email
1004	wiki	Washington	location
1004	wiki	U.S.	location
1004	wiki	North America	location
1004	wiki	United States	location
1005	wiki	State Government	organization
1005	wiki	58%	percentage
1006	wiki	San Antonio	location
1006	wiki	United States	location
1006	wiki	Texas	location
1007	wiki	Thomas	person

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Named Entity Finder Evaluator (ML Engine)

The NamedEntityFinderEvaluatorMap and NamedEntityFinderEvaluatorReduce functions operate as a row and a partition function, respectively. Each function takes a set of evaluating data and creates the precision, recall, and F-measure values of a specified maximum entropy data model. Neither function supports regular-expression-based or dictionary-based models.

Related information

Related information

Nondeterministic Results and UniqueID Syntax Element

## Named Entity Finder Evaluator Syntax

## NamedEntityFinderEvaluatorReduce version 1.5, NamedEntityFinderEvaluatorMap version 1.7

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON { table | view | (query) }
    USING
    TextColumn ('text_column')
    InputModelFile ('input_model_file')
  ) AS alias_1 PARTITION BY 1
) AS alias_2;
```

## Named Entity Finder Evaluator Syntax Elements

TextColumn

Specify the name of the input table column that contains the text to analyze.

InputModelFile

Specify name of the model file to evaluate.

## NamedEntityFinderEvaluatorMap Input

### Input Table Schema

Column	Data Type	Description
text_column	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax:  <START:entity_type> entity <END>  For example:  <START:location>Country1<END> has arrived

## NamedEntityFinderEvaluatorReduce Output

### Output Table Schema

Column	Data Type	Description
precision_val	INTEGER	Precision value of the model.
recall	DOUBLE PRECISION	Recall value of the model.
f_measure	DOUBLE PRECISION	F-measure (F1 score) of the model.

## Named Entity Finder Evaluator Example

### Input

- Input Table: nermem\_sports\_test, which has rows of sports news
- model\_file: location.sports, output by NamedEntityFinderTrainer Example

Input Table: nermem\_sports\_test

id	content
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .

id	content
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
14	Australian <START:PER> Tom Moody <END> took six for 82 but <START:PER> Chris Adams <END>
16	They were held up by a gritty 84 from <START:PER> Paul Johnson <END> but ex-England fast bowler <START:PER> Martin McCague <END> took four for 55 .
20	<START:LOCATION> LONDON <END> 1996-08-30
22	<START:LOCATION> Leicester <END> : <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs .
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON nermem_sports_test
    USING
    InputModelFile ('location.sports')
    TextColumn ('content')
  ) PARTITION BY 1
) AS dt;
```

## Output

```
precision_val    recall          f_measure
-----
0.847457627118644 0.7936507936507936 0.819672131147541
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Named Entity Finder Evaluator Syntax

### NamedEntityFinderEvaluatorReduce version 1.5, NamedEntityFinderEvaluatorMap version 1.7

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON { table | view | (query) }
    USING
    TextColumn ('text_column')
    InputModelFile ('input_model_file')
  ) AS alias_1 PARTITION BY 1
) AS alias_2;
```

## NamedEntityFinder Syntax

### Version 1.6

```
SELECT * FROM NamedEntityFinder (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON (configure_table) AS ConfigurationTable DIMENSION ]
  USING
  TextColumn ('text_column')
  [ Models ('entity_type[:model_type:{model_file|regular_expression}'][,...] | 'all' ) ]
  [ ShowContext ('context_words') ]
  [ EntityColName ('entity_column') ]
  [ Accumulate ( { 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related information

- Column Specification Syntax Elements
- Regular Expressions in Syntax Elements

## NamedEntityFinderTrainer Syntax

### Version 1.7

```
SELECT * FROM NamedEntityFinderTrainer (
  ON { table | view | (query) } PARTITION BY 1 [ ORDER BY order_column ]
  USING
  OutputModelFile (output_model_file)
  TextColumn ('text_column')
```

```

EntityType ('entity_type')
[ IterNum (iterator)]
[ Cutoff (cutoff)]
) AS alias;

```

For repeatable results, you must specify ORDER BY and order\_column must have a unique value for each row.

## NEREvaluator Syntax

### Version 1.9

```

SELECT * FROM NREvaluator (
  ON { table | view | (query) } PARTITION BY 1
  USING
  TextColumn ('text_column')
  ModelFile ('model_file[:jar_file]')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
) AS alias;

```

## NERExtractor Syntax

### Version 1.8

```

SELECT * FROM NRExtractor (
  ON input_table PARTITION BY { ANY | key }
  [ ON rules_table AS Rules DIMENSION ]
  [ ON dictionary_table AS Dict DIMENSION ]
  USING
  TextColumn ('text_column')
  [ InputModelFiles ('input_model_file[:jar_file]' [...]) ]
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ ShowContext ('n') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range } [...]) ]
) AS alias;

```

Related information

[Related information](#)  
[Column Specification Syntax Elements](#)

## NERTrainer Syntax

### Version 1.8

```

SELECT * FROM NERTrainer (
  ON { table | view | (query) } PARTITION BY 1

```

```

USING
ModelFileName (model_file)
TextColumn ('text_column')
[ ExtractorJAR ('jar_file') ]
FeatureTemplate ('template_file')
[ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
[ MaxIterNum (max_iteration_times) ]
[ Eta (eta_threshold_value) ]
[ MinOccurNum (threshold_value) ]
) AS alias;

```

## POSTagger Syntax

### Version 2.8

```

SELECT * FROM POSTagger (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')]
[ InputLanguage ({ 'en' | 'zh_Cn' }) ]
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;

```

Related information

[Related information](#)  
[Column Specification Syntax Elements](#)

## TextChunker Syntax

### Version 1.6

```

SELECT * FROM TextChunker (
  ON { table | view | (query) } PARTITION BY partition_key ORDER BY word_sn
  USING
  WordColumn ('word_column')
  POSColumn ('pos_tag_column')
) AS alias;

```

The input\_table is output table of the POSTagger (ML Engine) function, which contains the columns partition\_key and word\_sn.

## TextParser Syntax

### Version 1.14

```
SELECT * FROM TextParser (
  ON { table | view | (query) } [ PARTITION BY expression [,...] ]
  USING
  TextColumn ('text_column')
  [ ConvertToLowerCase ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemTokens ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Delimiter ('delimiter_regular_expression') ]
  [ OutputTotalWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Punctuation ('punctuation_regular_expression') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ TokenColName ('token_column') ]
  [ FrequencyColName ('frequency_column') ]
  [ TotalColName ('total_column') ]
  [ RemoveStopWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ PositionColName ('position_column') ]
  [ ListPositions ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ OutputByWord ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemExceptions ('exception_rule_file') ]
  [ StopWordsList ('stop_word_file') ]
) AS alias;
```

If you include the PARTITION BY clause, the function treats all rows in the same partition as a single document. If you omit the PARTITION BY clause, the function treats each row as a single document.

Related information

- Column Specification Syntax Elements
- Regular Expressions in Syntax Elements

## TextTagger Syntax

### Version 1.7

```
SELECT * FROM TextTagger (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON { table | view | (query) } AS Rules DIMENSION ]
  USING
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ TaggingRules ('rule AS tag' [,...]) ]
  [ Tokenize ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ OutputByTag ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ TagDelimiter ('delimiter') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related information

Related information

Column Specification Syntax Elements

## TFIDF Syntax

### TFIDF version 2.3, TF version 1.2

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  [ ON (SELECT COUNT (DISTINCT docid) FROM doccount_table) AS DocCount DIMENSION ]
  [ ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
  ]
  [ ON (SELECT DISTINCT (term) AS term, idf FROM tf_idf_output_table)
    AS IDF PARTITION BY term
  ]
) AS alias;
```

## Large Document Sets

For large documents sets, the DocPerTerm table is required.

For training, this is the syntax for large document sets:

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  ON (SELECT COUNT (DISTINCT docid) FROM doccount_table) AS DocCount DIMENSION
  ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
) AS alias ORDER BY docid;
```

For prediction, this is the syntax for large document sets:

```
SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
    [ USING Formula ({ 'normal' | 'bool' | 'log' | 'augment' }) ]
  ) AS TF PARTITION BY term
  [ ON (SELECT term, COUNT (DISTINCT docid) FROM docperterm_table GROUP BY term)
    AS DocPerTerm PARTITION BY term
  ]
]
```

```

[ ON (SELECT DISTINCT (term) AS term, idf FROM tf_idf_output_table)
  AS IDF PARTITION BY term
]
) AS alias ORDER BY docid;

```

## Small Document Sets

This syntax is acceptable for small document sets:

```

SELECT * FROM TFIDF (
  ON TF (
    ON { table | view | (query) } PARTITION BY docid
  ) AS TF PARTITION BY term
  ON (SELECT COUNT (DISTINCT docid) FROM input_table) AS DocCount DIMENSION
) AS alias ORDER BY docid;

```

# POSTagger (ML Engine)

The POSTagger function creates part-of-speech (POS) tags for the words in the input text. POS tagging is the first step in the syntactic analysis of a language, and an important preprocessing step in many natural language-processing applications.

The POSTagger function was developed on the Penn Treebank Project and Chinese Penn Treebank Project data set. Its POS tags comply with the tags defined by the two projects.

For the parts of speech used, see the following:

Text Language	Parts of Speech
English	<a href="https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html">https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html</a>
Chinese	<a href="https://www.sketchengine.co.uk/chinese-penn-treebank-part-of-speech-tagset/">https://www.sketchengine.co.uk/chinese-penn-treebank-part-of-speech-tagset/</a>

POSTagger uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## POSTagger Syntax

### Version 2.8

```
SELECT * FROM POSTagger (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')
  [ InputLanguage ({ 'en' | 'zh_Cn' }) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

#### Related Information:

[Column Specification Syntax Elements](#)

## POSTagger Syntax Elements

### TextColumn

Specify the name of the input column that contains the text to tag.

### InputLanguage

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese

**Accumulate**

[Optional] Specify the names of the input table columns to copy to the output table.

If you intend to use the POSTagger output table as input to the function [TextChunker \(ML Engine\)](#), then this syntax element must specify the input table columns that comprise the partition key.

## POSTagger Input

Table	Description						
Input table	Contains text to tag.						
Model table	Determined by InputLanguage syntax element: <table> <tr> <th>InputLanguage</th><th>Model File</th></tr> <tr> <td>English</td><td>pos_model_2.0_en_141008.bin</td></tr> <tr> <td>Simplified Chinese</td><td>pos_model_2.0_zh_cn_141008.bin</td></tr> </table> These model files are preinstalled on ML Engine.	InputLanguage	Model File	English	pos_model_2.0_en_141008.bin	Simplified Chinese	pos_model_2.0_zh_cn_141008.bin
InputLanguage	Model File						
English	pos_model_2.0_en_141008.bin						
Simplified Chinese	pos_model_2.0_zh_cn_141008.bin						

**Input Table Schema**

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>accumulate_column</i>	Any	Column to copy to output table.
<i>text_column</i>	VARCHAR	Text to tag. Each row of this column must contain a well-formatted sentence. To convert English text to formatted sentences, use <a href="#">SentenceExtractor (ML Engine)</a> function.

## POSTagger Output

**Output Table Schema**

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.



in statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. in other words, simple linear regression fits a straight line through the set of  $n$  points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible.

1

1 in IN

logistic regression was developed by statistician david cox in 1958[2][3] (although much work was done in the single independent variable case almost two decades earlier). the binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). as such it is not a classification method. it could be called a qualitative response/discrete choice model in the terminology of economics.

```
1      1 logistic      JJ
```

association rule learning is a method for discovering interesting relations between variables in large databases. it is intended to identify strong rules discovered in databases using different measures of interestingness. based on the concept of strong rules, rakesh agrawal et al.[2] introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (pos) systems in supermarkets. for example, the rule {onions, potatoes} $\Rightarrow$ {burger} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat.

1

```
1 association      NN
```

decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. it is one of the predictive modelling approaches used in statistics, data mining and machine learning. tree models where the target variable can take a finite set of values are called classification trees. in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. decision trees where the target variable can take continuous values (typically real numbers) are called regression

trees.

1 1 decision NN

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). it is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. cluster analysis itself is not one specific algorithm, but the general task to be solved. it can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. 1 1

cluster NN

cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). it is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. cluster analysis itself is not one specific algorithm, but the general task to be solved. it can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. 1 2

analysis NN

logistic regression was developed by statistician david cox in 1958[2][3] (although much work was done in the single independent variable case almost two decades earlier). the binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). as such it is not a classification method. it could be called a qualitative response/discrete choice model in the terminology of economics.

1 2 regression NN

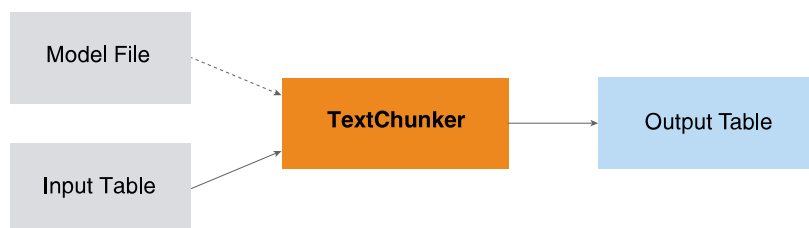
decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. it is one of the predictive modelling approaches used in statistics, data mining and machine learning. tree models where the target variable can take a finite set of values are called classification trees. in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

1 2 tree NN

association rule learning is a method for discovering interesting relations between variables in large databases. it is intended to identify strong rules

## TextChunker (ML Engine)

The TextChunker function divides text into phrases and assigns each phrase a tag that identifies its type.



*Text chunking* (also called *shallow parsing*) divides text into phrases in such a way that syntactically related words become members of the same phrase. Phrases do not overlap; that is, a word is a member of only one chunk.

For example, the sentence "He reckons the current account deficit will narrow to only # 1.8 billion in September ." can be divided as follows, with brackets delimiting phrases:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September]

After each opening bracket is a tag that identifies the chunk type (NP, VP, and so on). For information about chunk types, see [TextChunker Output](#).

For more information about text chunking, see:

- Erik F. Tjong Kim Sang and Sabine Buchholz, Introduction to the CoNLL-2000 Shared Task: Chunking. In: *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.
- Fei Sha and Fernando Pereira, Shallow Parsing with Conditional Random Fields. [2003]

TextChunker uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## TextChunker Syntax

### Version 1.6

```

SELECT * FROM TextChunker (
  ON { table | view | (query) } PARTITION BY partition_key ORDER BY word_sn
  USING
  WordColumn ('word_column')
  POSColumn ('pos_tag_column')
) AS alias;
  
```

The *input\_table* is output table of the [POSTagger \(ML Engine\)](#) function, which contains the columns *partition\_key* and *word\_sn*.

## TextChunker Syntax Elements

### WordColumn

Specify the name of the input table column that contains the words to chunk into phrases. Typically, this is the word column of the output table of the POSTagger function (described in [POSTagger Output](#)).

### POSColumn

Specify the name of the input table column the part-of-speech (POS) tag of words. Typically, this is the pos\_tag column of the output table of the POSTagger function (described in "POSTagger Output").

## TextChunker Input

Table	Description
Input table	<a href="#">POSTagger Output</a> table. When running POSTagger to create this table, specify in the Accumulate syntax element the name of the input column that contains the unique row identifiers.
Model file	chunker_default_model.bin, provided with function.

## TextChunker Output

### Output Table Schema

Column	Data Type	Description
partition_key	VARCHAR	Key of partition that contains text.
chunk_sn	INTEGER	Sequence number of phrase in sentence.
chunk	VARCHAR	Text chunk (syntactically related words).
chunk_tag	VARCHAR	Phrase type tag (see following table).

### Phrase Type Tags

Tag	Phrase Type
NP	noun phrase
VP	verb phrase
PP	prepositional phrase

Tag	Phrase Type
ADVP	adverb phrase
SBAR	subordinated clause
ADJP	adjective phrase
PRT	particles
CONJP	conjunction phrase
INTJ	interjection
LST	list marker
UCP	unlike coordinated phrase
O	punctuation marks

## TextChunker Examples

### TextChunker Example: POSTagger Output as Input

#### Input

- Input table: pos\_tmp, created by inputting the table cities to the POSTagger function

#### cities

paraid	paratext
1	I live in Los Angeles.
2	New York is a great city.
3	Chicago is a lot of fun, but the winters are very cold and windy.
4	Philadelphia and Boston have many historical sites.

This statement creates pos\_tmp:

```
CREATE multiset table pos_tmp AS (
  SELECT * FROM POSTagger (
    ON cities
    USING
    Accumulate ('paraid')
    TextColumn ('paratext')
  ) AS dt1
) WITH DATA;
```

### SQL Call

```
SELECT * FROM TextChunker (
  ON pos_tmp PARTITION BY paraid ORDER BY paraid, word_sn
  USING
  WordColumn ('word')
  POSColumn ('pos_tag')
) AS dt;
```

### Output

partition_key	chunk_sn	chunk	chunk_tag
1	1	i	NP
1	2	live	VP
1	3	in	PP
1	4	los angeles	NP
1	5	.	O
2	1	new york	NP
2	2	is	VP
2	3	a great city	NP
2	4	, filled	VP
2	5	with	PP
2	6	arts and culture	NP
2	7	.	O
3	1	chicago	NP
3	2	is	VP
3	3	a lot	NP
3	4	of	PP
3	5	fun	NP
3	6	,	O
3	7	but	O
3	8	the winters	NP
3	9	are	VP
3	10	very cold and windy	NP
3	11	.	O
4	1	philadelphia and boston	NP
4	2	have	VP
4	3	many historical sites	NP
4	4	.	O

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextChunker Example: SentenceExtractor and POSTagger Output as Input

### Input

paragraphs\_input

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of n points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible.
3	Logistic Regression	Logistic regression was developed by statistician David Cox in 1958[2][3] (although much work was done in the single independent variable case almost two decades earlier). The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). As such it is not a classification method. It could be called a qualitative response/discrete choice model in the terminology of economics.
4	Cluster analysis	Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. Cluster analysis itself is not one specific algorithm, but the general task to solve. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them.
5	Association rule learning	Association rule learning is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Rakesh Agrawal et al.[2] introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule {onions, potatoes} => {burger} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat.

SQL Call

TextChunker requires each sentence to have a unique identifier, and the input to TextChunker must be partitioned by that identifier.

```
SELECT * FROM TextChunker (  
  ON (  
    SELECT * FROM POSTagger (  
      ON (  
        SELECT paraid*1000+sentence_sn AS sentence_id, sentence FROM  
SentenceExtractor (  
      ON paragraphs_input  
      USING  
      TextColumn ('paratext')  
      Accumulate ('paraidd')  
    ) AS dt1  
  )  
  USING  
  TextColumn ('sentence')  
  Accumulate ('sentence_id')  
  ) AS dt2  
  ) PARTITION BY sentence_id ORDER BY word_sn  
  USING  
  WordColumn('word')  
  POSColumn('pos_tag')  
  ) AS dt;
```

Output

partition_key	chunk_sn	chunk	chunk_tag	
	1001	1	decision tree	
learning				NP
	1001	2		
uses				
		VP		
	1001	3	a decision	
tree				
NP				
	1001	4		
as				

		PP		
	1001	5	a predictive	
model				NP
	1001	6		
which				
		NP		
	1001	7		
maps				
		VP		
	1001	8		
observations				
		NP		
	1001	9		
about				
		PP		
	1001	10	an	
item				
		NP		
	1001	11		
to				
		PP		
	1001	12		
conclusions				
		NP		
	1001	13		
about				
		PP		
	1001	14	the items target	
value				NP
	1001			
15 .				
		0		
	1001	16		
it				
		NP		
	1001	17		
is				
		VP		
	1001	18		
one				
		NP		
	1001	19		
of				
		PP		

	1001	20 the predictive modelling	
approaches			NP
	1001	21	
used			
		VP	
	1001	22	
in			
		PP	
	1001	23 statistics , data mining and machine learning . tree	
models			NP
	1001	24	
where			
		ADVP	
	1001	25 the target	
variable			
NP			
	1001	26 can	
take			
VP			
	1001	27 a finite	
set			
NP			
	1001	28	
of			
		PP	
	1001	29	
values			
		NP	
	1001	30 are	
called			
VP			
	1001	31 classification	
trees			NP
	1001		
32 .			
		0	
	1001	33	
in			
		PP	
	1001	34 these tree	
structures			
NP			
	1001		
35 ,			

		0		
	1001	36		
leaves				
		VP		
	1001	37	represent class labels and	
branches				NP
	1001	38		
represent				
		VP		
	1001	39		
conjunctions				
		NP		
	1001	40		
of				
		PP		
	1001	41		
features				
		NP		
	1001	42		
that				
		NP		
	1001	43		
lead				
		VP		
	1001	44		
to				
		PP		
	1001	45	those class labels . decision	
trees				NP
	1001	46		
where				
		ADVP		
	1001	47	the target	
variable				
NP				
	1001	48	can	
take				
VP				
	1001	49	continuous	
values				
NP				
	1001	50	( typically real	
numbers				NP
	1001			

51 )			
		NP	
	1001	52 are	
called			
VP			
	1001	53 regression	
trees			
NP			
	1001		
54 .			
		0	
	2001	1	
in			
		PP	
	2001	2	
statistics			
		NP	
	2001		
3 ,			
		0	
	2001	4 simple linear	
regression			NP
	2001	5	
is			
		VP	
	2001	6 the least squares	
estimator			NP
	2001	7	
of			
		PP	
	2001	8 a linear regression	
model			NP
	2001	9	
with			
		PP	
	2001	10 a single explanatory	
variable .			NP
	2001	11	
in			
		PP	
	2001	12 other	
words			
NP			
	2001		

13 ,		0	
2001	14 simple linear		
regression			NP
2001	15		
fits			
	VP		
2001	16 a straight		
line			
NP			
2001	17		
through			
	PP		
2001	18 the		
set			
NP			
2001	19		
of			
	PP		
2001	20 n		
points			
	NP		
2001	21		
in			
	PP		
2001	22 such a		
way			
NP			
2001	23		
that			
	NP		
2001	24		
makes			
	VP		
2001	25 the		
sum			
NP			
2001	26		
of			
	PP		
2001	27 squared		
residuals			
NP			
2001	28		

of		PP
	2001	29 the model
(		NP
	2001	30
that		NP
	2001	31
is		VP
	2001	32 , vertical
distances		NP
	2001	33
between		PP
	2001	34 the
points		NP
	2001	35
of		PP
	2001	36 the
data		NP
	2001	37
set		VP
	2001	38
and		0
	2001	39 the fitted
line		NP
	2001	
40 )		VP
	2001	41 as
small		ADJP
	2001	42
as		PP

	2001	43		
possible				
		ADJP		
	2001			
44 .				
		0		
	3001	1 logistic		
regression				
NP				
	3001	2 was		
developed				
		VP		
	3001	3		
by				
		PP		
	3001	4 statistician david		
cox				NP
	3001	5		
in				
		PP		
	3001	6 1958[2][3](although much		
work				NP
	3001	7 was		
done				
VP				
	3001	8		
in				
		PP		
	3001	9 the single independent variable		
case				NP
	3001	10		
almost				
		ADVP		
	3001	11 two		
decades				
NP				
	3001	12		
earlier)				
		VP		
	3001			
13 .				
		0		
	3001	14 the binary logistic		
model				NP

estimate	3001	15 is used to	
VP			
probability	3001	16 the	
	NP		
of	3001	17	
	PP		
response	3001	18 a binary	
NP			
based	3001	19	
	VP		
on	3001	20	
	PP		
( features) .	3001	21 one or more predictor ( or independent ) variables	
	NP		
as	3001	22	
	PP		
such	3001	23	
	ADJP		
it	3001	24	
	NP		
is	3001	25	
	VP		
not	3001	26	
	0		
method	3001	27 a classification	
			NP
28 .	3001		
	VP		
it	3001	29	
	NP		
	3001	30 could be	

called			
VP			
	3001	31	a qualitative response/discrete choice
model			NP
	3001	32	
in			
		PP	
	3001	33	the
terminology			
		NP	
	3001	34	
of			
		PP	
	3001	35	
economics			
		NP	
	3001		
36 .			
		0	
	4001	1	cluster analysis or
clustering			NP
	4001	2	
is			
		VP	
	4001	3	the
task			
NP			
	4001	4	
of			
		PP	
	4001	5	
grouping			
		VP	
	4001	6	a
set			
		NP	
	4001	7	
of			
		PP	
	4001	8	
objects			
		NP	
	4001	9	
in			

		PP	
	4001	10	such a
way			
NP			
	4001	11	
that			
		NP	
	4001	12	
objects			
		VP	
	4001	13	
in			
		PP	
	4001	14	the same
group			
NP			
	4001	15	
( called			
		VP	
	4001	16	a
cluster )			
		NP	
	4001	17	
are			
		VP	
	4001	18	more
similar			
ADJP			
	4001	19	
(			
		0	
	4001	20	
in			
		PP	
	4001	21	some
sense			
NP			
	4001	22	
or			
		0	
	4001	23	
another )			
		NP	
	4001	24	

to		PP	
	4001	25	each
other			
NP			
	4001	26	
than			
		PP	
	4001	27	
to			
		PP	
	4001	28	
those			
		NP	
	4001	29	
in			
		PP	
	4001	30	other
groups			
NP			
	4001	31	
( clusters)			
		NP	
	4001		
32 .			
		0	
	4001	33	
it			
		NP	
	4001	34	
is			
		VP	
	4001	35	a main
task			
NP			
	4001	36	
of			
		PP	
	4001	37	exploratory data
mining			
			NP
	4001		
38 ,			
		0	
	4001	39	

and		0	
	4001	40	a common
technique			
NP			
	4001	41	
for			
		PP	
	4001	42	statistical data
analysis			
	4001	43	,
used			
		VP	
	4001	44	
in			
		PP	
	4001	45	many
fields			
NP			
	4001		
46 ,			
		0	
	4001	47	
including			
		PP	
	4001	48	machine
learning			
NP			
	4001		
49 ,			
		0	
	4001	50	pattern recognition , image analysis , information
retrieval , and bioinformatics . cluster analysis			
	4001	51	
itself			
		NP	
	4001	52	
is			
		VP	
	4001	53	
not			
		0	
	4001	54	one specific
algorithm			

NP		
	4001	
55 ,		0
	4001	56
but		0
	4001	57 the general
task		
NP		
	4001	58 to be
solved		
VP		
	4001	
59 .		0
	4001	60
it		
		NP
	4001	61 can be
achieved		
VP		
	4001	62
by		
		PP
	4001	63 various
algorithms		
NP		
	4001	64
that		
		NP
	4001	65
differ		
		VP
	4001	66
significantly		
		ADVP
	4001	67
in		
		PP
	4001	68 their
notion		
NP		
	4001	69

of		PP		
	4001	70		
what		NP		
	4001	71		
constitutes		VP		
	4001	72 a		
cluster		NP		
	4001	73		
and		0		
	4001	74		
how		ADVP		
	4001	75 to efficiently		
find			VP	
	4001	76		
them		NP		
	4001			
77 .		0		
	5001	1 association rule		
learning			NP	
	5001	2		
is		VP		
	5001	3 a		
method		NP		
	5001	4		
for		PP		
	5001	5		
discovering		VP		
	5001	6 interesting		
relations				
NP				
	5001	7		
between				

		PP	
	5001	8	
variables			
		NP	
	5001	9	
in			
		PP	
	5001	10 large	
databases			
NP			
	5001		
11 .			
		0	
	5001	12	
it			
		NP	
	5001	13 is intended to	
identify			VP
	5001	14 strong	
rules			
NP			
	5001	15	
discovered			
		VP	
	5001	16	
in			
		PP	
	5001	17	
databases			
		NP	
	5001	18	
using			
		VP	
	5001	19 different	
measures			
NP			
	5001	20	
of			
		PP	
	5001	21	
interestingness			
		NP	
	5001	22 .	
based			

		VP	
	5001	23	
on			
		PP	
	5001	24 the	
concept			
NP			
	5001	25	
of			
		PP	
	5001	26 strong	
rules			
NP			
	5001		
27 ,			
		0	
	5001	28 rakesh agrawal et al.[2 ] introduced association	
rules			NP
	5001	29	
for			
		PP	
	5001	30 discovering	
regularities			
NP			
	5001	31	
between			
		PP	
	5001	32	
products			
		NP	
	5001	33	
in			
		PP	
	5001	34 large-scale transaction	
data			NP
	5001	35	
recorded			
		VP	
	5001	36	
by			
		PP	
	5001	37 point-of-sale ( pos )	
systems			NP
	5001	38	

in		PP	
	5001	39	
supermarkets		NP	
	5001		
40 .		0	
	5001	41	
for		PP	
	5001	42	
example		NP	
	5001		
43 ,		0	
	5001	44 the rule { onions ,	
potatoes}=>{burger			NP
	5001	45 }	
found		VP	
	5001	46	
in		PP	
	5001	47 the sales	
data			
NP			
	5001	48	
of		PP	
	5001	49 a	
supermarket		NP	
	5001	50 would	
indicate			
VP			
	5001	51	
that		SBAR	
	5001	52	
if		SBAR	
	5001	53 a	

customer		
	NP	
5001	54	
buys		
	VP	
5001	55	
onions		
	NP	
5001	56	
and		
	0	
5001	57	
potatoes		
	VP	
5001	58	
together		
	ADVP	
5001		
59 ,		
	0	
5001	60	
they		
	NP	
5001	61	
are		
	VP	
5001	62	
likely		
	ADJP	
5001	63 to also	
buy		
VP		
5001	64 hamburger	
meat		
NP		
5001		
65 .		
	0	

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## TextParser (ML Engine)

The TextParser function tokenizes an input stream of words, optionally *stems* them (reduces them to their root forms), and then outputs them. The function can either output all words in one row or output each word in its own row with (optionally) the number of times that the word appears.

The TextParser function uses Porter2 as the stemming algorithm.

The TextParser function reads a document into a memory buffer and creates a hash table. The dictionary for the document must not exceed available memory; however, a million-word dictionary with an average word length of ten bytes requires only 10 MB of memory.

TextParser uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

### TextParser Syntax

#### Version 1.14

```
SELECT * FROM TextParser (
  ON { table | view | (query) } [ PARTITION BY expression [,...] ]
  USING
  TextColumn ('text_column')
  [ ConvertToLowerCase ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemTokens ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Delimiter ('delimiter_regular_expression') ]
  [ OutputTotalWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Punctuation ('punctuation_regular_expression') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ TokenColName ('token_column') ]
  [ FrequencyColName ('frequency_column') ]
  [ TotalColName ('total_column') ]
  [ RemoveStopWords ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ PositionColName ('position_column') ]
  [ ListPositions ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ OutputByWord ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ StemExceptions ('exception_rule_file') ]
  [ StopWordsList ('stop_word_file') ]
) AS alias;
```

If you include the PARTITION BY clause, the function treats all rows in the same partition as a single document. If you omit the PARTITION BY clause, the function treats each row as a single document.

**Related Information:**

[Column Specification Syntax Elements](#)

[Regular Expressions in Syntax Elements](#)

## TextParser Syntax Elements

### TextColumn

Specify the name of the input column with contents to tokenize.

### ConvertToLowerCase

[Optional] Specify whether to convert input text to lowercase.

The function ignores this syntax element if the StemTokens syntax element has the value 'true'.

Default: 'true'

### StemTokens

[Optional] Specify whether to stem the tokens—that is, whether to apply the Porter2 stemming algorithm to each token to reduce it to its root form. Before stemming, the function converts the input text to lowercase and applies the RemoveStopWords syntax element.

Default: 'false'

### Delimiter

[Optional] Specify a regular expression that represents the word delimiter.

The function uses only specified characters as delimiters. For example, if you specify Delimiter ('-'), the function uses only the hyphen character as a delimiter. To use the hyphen and the default delimiters, specify Delimiter ('[- \\t\\f\\r\\n]+').

Default: '[- \\t\\f\\r\\n]+'

### OutputTotalWords

[Optional] Specify whether to output a column that contains the total number of words in the input document.

Default: 'false'

### Punctuation

[Optional] Specify a regular expression that represents the punctuation characters to remove from the input text. With StemTokens ('true'), the recommended value is '[\\[\\[ ., ? \\ ! : ; ~ ( ) \\ \\ ] ] + '.

Default: '[ . , ! ? ] '

### Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

No *accumulate\_column* can be the same as *token\_column* or *total\_column*.

Default: All input columns

#### **TokenColName**

[Optional] Specify the name of the output column that contains the tokens.

Default: 'token'

#### **FrequencyColName**

[Optional] Specify the name of the output column that contains the frequency of each token.

The function ignores this syntax element if the OutputByWord syntax element has the value 'false'.

Default: 'frequency'

#### **TotalColName**

[Optional] Specify the name of the output column that contains the total number of words in the input document.

Default: 'total\_count'

#### **RemoveStopWords**

[Optional] Specify whether to remove stop words from the input text before parsing.

Default: 'false'

#### **PositionColName**

[Optional] Specify the name of the output column that contains the position of a word within a document.

Default: 'location'

#### **ListPositions**

[Optional] Specify whether to output the position of a word in list form.

The function ignores this syntax element if the OutputByWord syntax element has the value 'false'.

Default: 'false' (The function outputs a row for each occurrence of the word.)

#### **OutputByWord**

[Optional] Specify whether to output each token of each input document in its own row in the output table. If you specify 'false', then the function outputs each tokenized input document in one row of the output table.

Default: 'true'

#### **StemExceptions**

[Optional] Specify the location of the file that contains the stemming exceptions. A stemming exception is a word followed by its stemmed form. The word and its stemmed form are separated by white space. Each stemming exception is on its own line in the file. For example:

```
bias bias
news news
goods goods
```

```

lying lie
ugly ugly
sky sky
early earli

```

The words 'lying', 'ugly', and 'early' are to become 'lie', 'ugli', and 'earli', respectively. The other words are not to change.

Default: No stemming exceptions

### StopWordsList

[Optional] Specify the location of the file that contains the stop words (words to ignore when parsing text). Each stop word is on its own line in the file. For example:

```

a
an
the
and
this
with
but
will

```

Default: No stop words

## TextParser Input

If you include the PARTITION BY clause, the function treats all rows in the same partition as a single document. If you omit the PARTITION BY clause, the function treats each row as a single document.

### Input Table Schema

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to parse.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

## TextParser Output

The output table schema depends on the OutputByWord syntax element.

### Output Table Schema, Output\_By\_Word ('true') (Default)

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.

Column	Data Type	Description
<i>token_column</i>	CLOB	Token.
<i>frequency_column</i>	INTEGER	Frequency of token.
<i>position_column</i>	VARCHAR	Position of word within document.

### Output Table Schema, Output\_By\_Word ('false')

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.
<i>token_column</i>	CLOB	Token.

## TextParser Examples

### TextParser Example: StopWordsList, No StemExceptions

#### Input

- InputTable: complaints, a log of vehicle complaints.  
The category column indicates whether the vehicle was in a crash.
- Stop words file: stopwords.txt, which is preinstalled on ML Engine (shown in [TextClassifierTrainer Example](#))

#### complaints

doc_id	text_data	category
1	consumer was driving approximately 45 mph hit a deer with the front bumper and then ran into an embankment head-on passenger's side air bag did deploy hit windshield and deployed outward. driver's side airbag cover opened but did not inflate it was still folded causing injuries.	crash
2	when vehicle was involved in a crash totalling vehicle driver's side/ passenger's side air bags did not deploy. vehicle was making a left turn and was hit by a ford f350 traveling about 35 mph on the front passenger's side. driver hit his head-on the steering wheel. hurt his knee and received neck and back injuries.	crash
3	consumer has experienced following problems; 1.) both lower ball joints wear out excessively; 2.) head gasket leaks; and 3.) cruise control would shut itself off while driving without foot pressing on brake pedal.	no_crash
...	...	...

**SQL Call**

```

SELECT * FROM TextParser (
  ON complaints
  USING
  TextColumn ('text_data')
  ConvertToLowerCase ('true')
  StemTokens ('false')
  OutputByWord ('true')
  Punctuation ('\[,?\!\\\')
  RemoveStopWords ('true')
  ListPositions ('true')
  Accumulate ('doc_id', 'category')
  StopWordsList ('stopwords.txt')
) AS dt ORDER BY doc_id,category,token,frequency,location;

```

**Output**

doc_id	category	token	frequency	location
1	crash	45	1	4
1	crash	air	1	22
1	crash	airbag	1	33
1	crash	approximately	1	3
1	crash	bag	1	23
1	crash	bumper	1	12
1	crash	causing	1	44
1	crash	consumer	1	0
1	crash	cover	1	34
1	crash	deer	1	8
1	crash	deploy	1	25
1	crash	deployed	1	29
1	crash	did	2	24,37
1	crash	driver's	1	31
1	crash	driving	1	2
1	crash	embankment	1	18
1	crash	folded	1	43
1	crash	front	1	11
1	crash	head-on	1	19
1	crash	hit	2	6,26
1	crash	inflate	1	39
1	crash	injuries	1	45
1	crash	it	1	40
1	crash	mph	1	5

1 crash	not	1 38
1 crash	opened	1 35
1 crash	outward	1 30
1 crash	passenger's	1 20
1 crash	ran	1 15
1 crash	side	2 21,32
1 crash	still	1 42
1 crash	then	1 14
1 crash	windshield	1 27
2 crash	35	1 33
2 crash	about	1 32
2 crash	air	1 13
2 crash	back	1 54
2 crash	bags	1 14
2 crash	by	1 27
2 crash	crash	1 6
2 crash	deploy	1 17
2 crash	did	1 15
2 crash	driver	1 40
2 crash	driver's	1 9
2 crash	f350	1 30
2 crash	ford	1 29
2 crash	front	1 37
2 crash	head-on	1 43
2 crash	his	2 42,48
2 crash	hit	2 26,41
2 crash	hurt	1 47
2 crash	injuries	1 55
2 crash	involved	1 3
2 crash	knee	1 49
2 crash	left	1 22
2 crash	making	1 20
2 crash	mph	1 34
2 crash	neck	1 52
2 crash	not	1 16
2 crash	on	1 35
2 crash	passenger's	2 11,38
2 crash	received	1 51
2 crash	side	2 12,39
2 crash	side/	1 10
2 crash	steering	1 45
2 crash	totalling	1 7
2 crash	traveling	1 31
2 crash	turn	1 23

2 crash	vehicle	3 1,8,18
2 crash	wheel	1 46
2 crash	when	1 0
3 no_crash	1)	1 5
3 no_crash	2)	1 13
3 no_crash	3)	1 18
3 no_crash	ball	1 8
3 no_crash	both	1 6
3 no_crash	brake	1 31
3 no_crash	consumer	1 0
3 no_crash	control	1 20
3 no_crash	cruise	1 19
3 no_crash	driving	1 26
3 no_crash	excessively;	1 12
3 no_crash	experienced	1 2
3 no_crash	following	1 3
3 no_crash	foot	1 28
3 no_crash	gasket	1 15
3 no_crash	has	1 1
3 no_crash	head	1 14
3 no_crash	itself	1 23
3 no_crash	joints	1 9
3 no_crash	leaks;	1 16
3 no_crash	lower	1 7
3 no_crash	off	1 24
3 no_crash	on	1 30
3 no_crash	out	1 11
3 no_crash	pedal	1 32
3 no_crash	pressing	1 29
3 no_crash	problems;	1 4
3 no_crash	shut	1 22
3 no_crash	wear	1 10
3 no_crash	while	1 25
3 no_crash	without	1 27
3 no_crash	would	1 21
4 no_crash	after	1 6
4 no_crash	back	1 18
4 no_crash	been	1 40
4 no_crash	case	2 1,36
4 no_crash	completed	1 10
4 no_crash	consumer	1 15
4 no_crash	dealer	2 20,22
4 no_crash	driveshaft	1 31
4 no_crash	has	1 39

# Named Entity Recognition (NER) Functions (ML Engine)

*Named entity recognition (NER)* is a process for finding specified entities in text. For example, a simple news named-entity recognizer for English might find the person "John J. Smith" and the location "Seattle" in the text string "John J. Smith lives in Seattle."

NER functions let you specify how to extract named entities when training the data models. ML Engine provides two sets of NER functions:

Function Set	Supported Languages
<a href="#">NER Functions (CRF Model Implementation)</a>	English, simplified Chinese, traditional Chinese
<a href="#">NER Functions (Maximum Entropy Model Implementation)</a>	English

## NER Functions (CRF Model Implementation)

Function	Description
<a href="#">NERTrainer</a>	Takes training data and outputs CRF model (binary file).
<a href="#">NERExtractor</a>	Takes input documents and extracts specified entities, using one or more CRF models and, if appropriate, rules (regular expressions) or a dictionary. Uses models to extract names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and dictionary to extract known entities.
<a href="#">NEREvaluator</a>	Evaluates CRF model.

The CRF model implementation supports English, simplified Chinese, and traditional Chinese text.

### Related Information:

[NER Functions \(Maximum Entropy Model Implementation\)](#)

## NERTrainer

The NERTrainer function takes training data and outputs a CRF model (a binary file) that can be specified in the function [NERExtractor](#) and [NEREvaluator](#).

NERTrainer uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NERTrainer Syntax

**Version 1.8**

```

SELECT * FROM NERTrainer (
  ON { table | view | (query) } PARTITION BY 1
  USING
  ModelFileName (model_file)
  TextColumn ('text_column')
  [ ExtractorJAR ('jar_file') ]
  FeatureTemplate ('template_file')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ MaxIterNum (max_iteration_times) ]
  [ Eta (eta_threshold_value) ]
  [ MinOccurNum (threshold_value) ]
) AS alias;

```

**NERTrainer Syntax Elements****ModelFileName**

Specify the name of the model file that the function creates and installs on ML Engine.

**TextColumn**

Specify the name of the input table column that contains the text to analyze.

**ExtractorJAR**

[Optional] Specify the name of the JAR file that contains the Java classes that extract features. You must install this JAR file on ML Engine before calling the function.

The name *jar\_file* is case-sensitive.

ML Engine does not support the creation of new extractor classes. However, it does support existing JAR files—for installation instructions, see *Teradata Vantage™ User Guide*, B700-4002.

Default behavior: The function uses only the predefined extractor classes.

**FeatureTemplate**

Specify the name of the file that specifies how to create features when training the model.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**MaxIterNum**

[Optional] Specify the maximum number of iterations.

Default: 1000

**Eta**

[Optional] Specify the tolerance of the termination criterion. Defines the differences of the values of the loss function between two sequential epochs.

When training a model, the function performs  $n$ -times iterations. At the end of each epoch, the function calculates the loss or cost function on the training samples. If the loss function value change is very small between two sequential epochs, the function considers the training process to have converged.

The function defines Eta as:

$$\text{Eta} = (f(n) - f(n-1)) / f(n-1)$$

where  $f(n)$  is the loss function value of the  $n$ th epoch.

Default: 0.0001

**MinOccurNum**

[Optional] Specify the minimum number times that a feature must occur in the input text before the function uses the feature to construct the model.

Default: 0

**NERTrainer Input****Input Table Schema**

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Text to analyze. Within text, each entity must be identified with this syntax: <code>&lt;START:entity_type&gt;entity&lt;END&gt;</code> For example: <code>&lt;START:location&gt;Country1&lt;END&gt; has arrived</code>

**NERTrainer Output**

The function outputs a message and a CRF model (a binary file installed on ML Engine).

## Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Reports training time and file size of model.

## NERTrainer Example

### Input

- Input table: ner\_sports\_train, a collection of sports news items (500 rows)
- Feature template file: template\_1.txt, which is preinstalled on ML Engine.

#### ner\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOC> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOC> Grace Road <END>
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
...	...

### SQL Call

```
SELECT * FROM NERTrainer (
  ON ner_sports_train PARTITION BY 1
  USING
    TextColumn ('content')
    FeatureTemplate ('template_1.txt')
    OutputModelFile ('ner_model.bin')
) AS dt;
```

## Output

```
train_result
-----
Model generated.
Training time(s): 3.129
File size(KB): 374
Model successfully installed.
```

The model file, ner\_model.bin, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NERExtractor

The NERExtractor function takes input documents and extracts specified entities, using one or more CRF models (output by the function [NERTrainer](#)) and, if appropriate, rules (regular expressions) or a dictionary.

The function uses models to extract the names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and a dictionary to extract known entities.

NERExtractor uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NERExtractor Syntax

### Version 1.8

```
SELECT * FROM NERExtractor (
  ON input_table PARTITION BY { ANY | key }
  [ ON rules_table AS Rules DIMENSION ]
  [ ON dictionary_table AS Dict DIMENSION ]
  USING
  TextColumn ('text_column')
  [ InputModelFiles ('input_model_file[:jar_file]' [,...]) ]
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ ShowContext ('n') ]
```

```
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

**Related Information:**

[Column Specification Syntax Elements](#)

**NERExtractor Syntax Elements****TextColumn**

Specify the name of the input table column that contains the text to analyze.

**InputModelFiles**

[Optional] Specify the CRF models (binary files) to use, output by [NERTrainer](#). If you specified the ExtractorJAR syntax element in the NERTrainer call that created *input\_model\_file*, then you must specify the same *jar\_file* in this syntax element. You must install *input\_model\_file* and *jar\_file* in ML Engine before calling the NERExtractor function.

The names *input\_model\_file* and *jar\_file* are case-sensitive.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**ShowContext**

[Optional] Specify the number of context words to output (a positive integer). The function outputs the *n* words that precede the entity, the entity, and the *n* words that follow the entity.

Default: 0

**Accumulate**

[Optional] Specify the names of the input table columns to copy to the output table.

**NERExtractor Input**

Table	Description
Input table	Text to analyze. <b>Tip:</b> To optimize function performance, remove punctuation marks from text with <a href="#">TextParser (ML Engine)</a> function.
Rules	[Optional] Rules to use when extracting entities from text.

Table	Description
Dict	[Optional] Dictionary to use when extracting entities from text.

### Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze.
<i>accumulate_column</i>	Any	Column to copy to output table.

### Rules Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
regex	VARCHAR	Regular expression that represents an entity of this type. Expression must conform to Java Regex standard, documented at <a href="http://docs.oracle.com/javase/tutorial/essential/regex/quant.html">http://docs.oracle.com/javase/tutorial/essential/regex/quant.html</a> .

### Dict Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
dict	VARCHAR	Dictionary word.

## NERExtractor Output

### Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
sn	INTEGER	Serial number of extracted entity.
entity	VARCHAR	Extracted entity.
type	VARCHAR	Type of extracted entity.
start	INTEGER	Start position of extracted entity in input text.
end	INTEGER	End position of extracted entity in input text.

Column	Data Type	Description
context	VARCHAR	[Column appears only with ShowContent syntax element.] Context of extracted entity.
approach	VARCHAR	Method used to identify extracted entity—CRF, RULE, or DICT.

## NERExtractor Example

### Input

- Input table: ner\_sports\_test2, which contains text to analyze.
- Rules: rule\_table, which is preinstalled on ML Engine.
- Model: ner\_model.bin, output by [NERTrainer Example](#).

#### Input table: ner\_sports\_test2

id	content
528	email sports@espn.com to contact for all sport info
529	email cricket@espn.com to contact for all cricket info
530	email tennis@espn.com to contact for all tennis info
531	1= Igor Trandenkov (Russia) 5.86
532	3. Maksim Tarasov (Russia) 5.86
533	4. Tim Lobinger (Germany) 5.80
534	5. Igor Potapovich (Kazakstan) 5.80
535	6. Jean Galfione (France) 5.65
536	7. Pyotr Bochkary (Russia) 5.65
537	8. Dmitri Markov (Belarus) 5.65
583	GENEVA 1996-08-30
584	UEFA came down heavily on Belgian club Standard Liege on Friday for disgraceful behaviour in an Intertoto final match against Karlsruhe of Germany .
585	The Belgian club were fined 25
586	He was sent off for insulting the referee and then urged his team mates to protest .
587	Roberto Bisconti will be sidelined for six Euro ties after pushing the referee in the back as he protested about a Karlsruhe goal
588	Karlsruhe won the August 20 match 3-1 thanks to two late goals .

id	content
589	They took the tie 3-2 on aggregate and qualified for the UEFA Cup .
591	ATHLETICS - HARRISON
592	MONTE CARLO 1996-08-30
593	Olympic champion Kenny Harrison and world record holder Jonathan Edwards will both take part in a triple jump competition at the Solidarity Meeting for Sarajevo on September 9 .
594	The International Amateur Athletic Federation said on Friday that a schedule reshuffle had allowed organisers to hold a men s triple jump as well as the women s long jump on the one usable runway at the war-devastated Kosevo stadium .
595	Atlanta Games silver medal winner Edwards has called on other leading athletes to take part in the Sarajevo meeting -- a goodwill gesture towards Bosnia as it recovers from the war in the Balkans -- two days after the grand prix final in Milan .
596	Edwards was quoted as saying : What type of character do we show by going to the IAAF Grand Prix Final in Milan where there is a lot of money to make but refusing to make the trip to Sarajevo as a humanitarian gesture ?
598	SOCCER - BARATELLI TO COACH NICE .
599	NICE
600	Former international goalkeeper Dominique Baratelli is to coach struggling French first division side Nice
601	Baratelli
602	Nice have been unable to win any of their four league matches played this season and are lying a lowly 18th in the table .

**Rules: rule\_table**

type	regex
email	<code>[w\~]([\.w])+[w]+@[([w\~]+\.)+[a-zA-Z]{2,4}</code>

**SQL Call**

```

SELECT * FROM NERExtractor (
  ON ner_sports_test2 PARTITION BY ANY
  ON rule_table AS Rules DIMENSION
  USING
    TextColumn ('content')
    InputModelFiles ('ner_model.bin')
    ShowContext (2)
    Accumulate ('id')
) AS dt ORDER BY id, sn;

```

**Output**

id	sn	entity	type_ner	start_ner	end_ner	approach
-----						
528	1	sports@espn.com	email	2	2	... email sports@espn.com
		to contact	RULE			
529	1	cricket@espn.com	email	2	2	... email cricket@espn.com
		to contact	RULE			
530	1	tennis@espn.com	email	2	2	... email tennis@espn.com
		to contact	RULE			
531	1	Igor Trandankov	PER	2	3	... 1= Igor Trandankov
		(Russia) 5.86	CRF			
532	1	Maksim Tarasov	PER	2	3	... 3. Maksim Tarasov
		(Russia) 5.86	CRF			
533	1	Tim Lobinger	PER	2	3	... 4. Tim Lobinger
		(Germany) 5.80	CRF			
534	1	Igor Potapovich	PER	2	3	... 5. Igor Potapovich
		(Kazakstan) 5.80	CRF			
535	1	Jean Galfione	PER	2	3	... 6. Jean Galfione
		(France) 5.65	CRF			
536	1	Pyotr Bochkary	PER	2	3	... 7. Pyotr Bochkary
		(Russia) 5.65	CRF			
537	1	Dmitri Markov	PER	2	3	... 8. Dmitri Markov
		(Belarus) 5.65	CRF			
583	1	GENEVA	LOC	1	1	... ... GENEVA
		1996-08-30 ...	CRF			
584	1	Standard Liege	PER	8	9	Belgian club Standard
		Liege on Friday	CRF			
587	1	Roberto Bisconti	PER	1	2	... ... Roberto Bisconti
		will be	CRF			
591	1	HARRISON	PER	3	3	ATHLETICS -
		HARRISON ... ..	CRF			
592	1	MONTE CARLO	PER	1	2	... ... MONTE CARLO
		1996-08-30 ...	CRF			
593	1	Kenny Harrison	PER	3	4	Olympic champion Kenny
		Harrison and world	CRF			
593	2	Jonathan Edwards	PER	9	10	record holder Jonathan
		Edwards will both	CRF			
596	1	What	ORG	7	7	saying : What type
		of	CRF			
598	1	BARATELLI TO	PER	3	4	SOCCER - BARATELLI TO
		COACH NICE	CRF			

599	1	NICE	PER	1	1 ... ..
NICE	...	...		CRF	
600	1	Dominique Baratelli	PER	4	5 international goalkeeper
Dominique Baratelli	is	to	CRF		
600	2	Nice	PER	14	14 division side
Nice	...	...		CRF	
601	1	Baratelli	PER	1	1 ... ..
Baratelli	...	...		CRF	

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NEREvaluator

The NREvaluator function evaluates a CRF model (output by the function [NERTrainer](#)).

NEREvaluator uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NEREvaluator Syntax

### Version 1.9

```
SELECT * FROM NREvaluator (
  ON { table | view | (query) } PARTITION BY 1
  USING
  TextColumn ('text_column')
  ModelFile ('model_file[:jar_file]')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
) AS alias;
```

## NEREvaluator Syntax Elements

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### ModelFile

Specify the CRF model file to evaluate, created and automatically installed by [NERTrainer](#).

If you specified the ExtractorJAR syntax element in the NERTrainer call that created *model\_file*, then you must specify the same *jar\_file* in this syntax element. You must install the *jar\_file* on ML Engine before calling the NREvaluator function.

The names *model\_file* and *jar\_file* are case-sensitive.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**NEREvaluator Input**

The input table has the same schema as the [NERExtractor Input](#) table.

**NEREvaluator Output****Output Table Schema**

Column	Data Type	Description
type	VARCHAR	Entity type. Final row value: -AVG-
precision	DOUBLE PRECISION	Precision value of the entity type. Final row value: Average precision value for all entity types.
recall	DOUBLE PRECISION	Recall value of the entity type. Final row value: Average recall value for all entity types.
f1_measure	DOUBLE PRECISION	F <sub>1</sub> score (F-measure) of the entity type. Final row value: Average F <sub>1</sub> score for all entity types.

**NEREvaluator Example**

This function evaluates the efficacy of the model file `ner_model.bin`, created by the `NERTrainer` function in terms of precision, recall, and `f1_measure`.

**Input**

- `ner_model.bin`, output by [NERTrainer Example](#)

**SQL Call**

```
SELECT * FROM NREvaluator (
  ON ner_sports_test2 PARTITION BY 1
```

```

USING
  TextColumn ('content')
  ModelFile ('ner_model.bin')
) AS dt;

```

## Output

```

type_ner precision_ner recall f1_measure
-----
LOC              1 0.4444      0.6154
ORG              0      0      -1
PER             0.7222 0.8125      0.7647
-AVG-           0.7778 0.4884      0.6

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NER Functions (Maximum Entropy Model Implementation)

Function	Description
<a href="#">NamedEntityFinderTrainer</a>	Takes training data and outputs a maximum entropy model (binary file).
<a href="#">NamedEntityFinder</a>	Evaluates input, identifies tokens based on specified model, and outputs tokens with detailed information. Uses model to extract entity types 'PERSON', 'LOCATION', and 'ORGANIZATION' and rules to extract entity types 'DATE', 'TIME', 'EMAIL' and 'MONEY'. If you specify these entity names, the function invokes the default model types and model file names. To extract all entities in one NamedEntityFinder call, specify 'ALL'.
<a href="#">Named Entity Finder Evaluator</a>	Evaluates maximum entropy model.

The maximum entropy model implementation supports only English text.

### Related Information:

[NER Functions \(CRF Model Implementation\)](#)

## NamedEntityFinderTrainer

The NamedEntityFinderTrainer function takes training data and outputs a Maximum Entropy data model. The function is based on OpenNLP, and follows its annotation. For more information on OpenNLP, see <https://opennlp.apache.org/docs/1.8.4/manual/opennlp.html>.

The trainer supports only the English language.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NamedEntityFinderTrainer Syntax

### Version 1.7

```
SELECT * FROM NamedEntityFinderTrainer (
  ON { table | view | (query) } PARTITION BY 1 [ ORDER BY order_column ]
  USING
  OutputModelFile (output_model_file)
  TextColumn ('text_column')
  EntityType ('entity_type')
  [ IterNum (iterator)]
  [ Cutoff (cutoff)]
) AS alias;
```

For repeatable results, you must specify ORDER BY and *order\_column* must have a unique value for each row.

## NamedEntityFinderTrainer Syntax Elements

### OutputModelFile

Specify the name of the data model file to create.

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### EntityType

Specify the entity type to train (for example, PERSON). The input training documents must contain the same tag.

### IterNum

[Optional] Specify the iterator number for training (an openNLP training parameter).

Default: 100

### Cutoff

[Optional] Specify the cutoff number for training (an openNLP training parameter).

Default: 5

## NamedEntityFinderTrainer Input

### Input Table Schema

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax: <div>&lt;START:entity_type&gt;entity&lt;END&gt;</div> For example: <div>&lt;START:location&gt;Country1&lt;END&gt; has arrived</div>

## NamedEntityFinderTrainer Output

The function outputs a message and a Max Entropy model (a binary file automatically installed on ML Engine).

### Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Message indicating whether the function ran successfully.

## NamedEntityFinderTrainer Example

### Input

- Input Table: nermem\_sports\_train, which has 50 rows of sports news

Input Table: nermem\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>

id	content
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderTrainer (
  ON nermem_sports_train PARTITION BY 1
  USING
    EntityType ('LOCATION')
    TextColumn ('content')
    OutputModelFile (location.sports)
) AS dt;
```

## Output

```
train_result
-----
model installed
```

The model table, location.sports, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NamedEntityFinder

The NamedEntityFinder function evaluates the input, identifies tokens based on the specified model, and outputs the tokens with detailed information. The function does not identify sentences; it simply tokenizes. Token identification is not case-sensitive.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NamedEntityFinder Syntax

**Version 1.6**

```

SELECT * FROM NamedEntityFinder (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON (configure_table) AS ConfigurationTable DIMENSION ]
  USING
  TextColumn ('text_column')
  [ Models ('entity_type[:model_type:{model_file|regular_expression}'][,...] |
  'all' }) ]
  [ ShowContext ('context_words') ]
  [ EntityColName ('entity_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;

```

**Related Information:**

[Column Specification Syntax Elements](#)

[Regular Expressions in Syntax Elements](#)

**NamedEntityFinder Syntax Elements****TextColumn**

Specify the name of the input table column that contains the text to analyze.

**Models**

[Optional] Required if you do not specify ConfigurationTable, in which case you cannot specify 'all'. Specify the model items to load.

If you specify both ConfigurationTable and this syntax element, the function loads the specified model items from ConfigurationTable.

The *entity\_type* is the name of an entity type (for example, PERSON, LOCATION, or EMAIL), which appears in the output table.

<i>model_type</i>	Description
'max entropy'	Maximum entropy language model output by training.
'rule'	Rule-based model, a plain text file with one regular expression on each line.
'dictionary'	Dictionary-based model, a plain text file with one word on each line.
'reg exp'	Regular expression that describes <i>entity_type</i> .

If *model\_type* is 'reg exp', specify *regular\_expression* (a regular expression that describes *entity\_type*); otherwise, specify *model\_file* (the name of the model file).

If you specify ConfigurationTable, you can use *entity\_type* as a shortcut. For example, if the ConfigurationTable has the row 'organization, max entropy, en-ner-organization.bin', you can

specify Models ('organization') as a shortcut for Models ('organization:max entropy:en-ner-organization.bin').

---

**Note:**

For *model\_type* 'max entropy', if you specify ConfigurationTable and omit this syntax element, then the JVM of the worker node needs more than 2GB of memory.

---

Default: 'all' (If you specify ConfigurationTable but omit this syntax element.)

**ShowContext**

[Optional] Specify the number of context words to output. If *context\_words* is *n* (which must be a positive integer), the function outputs the *n* words that precede the entity, the entity, and the *n* words that follow the entity.

Default: 0

**EntityColName**

[Optional] Specify the name of the output table column that contains the entity names.

Default: 'entity'

**Accumulate**

[Optional] Specify the names of input columns to copy to the output table. No *accumulate\_column* can be an *entity\_column*.

Default: All input columns

## Creating the Table of Default Models

Before calling the NamedEntityFinder function, you must create the table of default models. To create the table, use this command:

```
DROP TABLE nameFind_configure;

CREATE MULTISET TABLE nameFind_configure (
  model_name VARCHAR(50),
  model_type VARCHAR(50),
  model_file VARCHAR(50)
);
```

Default English-language models are provided with the SQL functions. Before using these models, you must create a default *configure\_table*, as follows:

```
INSERT INTO nameFind_configure VALUES ('person','max entropy','en-ner-person.bin');
INSERT INTO nameFind_configure VALUES ('location','max entropy','en-ner-location.bin');
INSERT INTO nameFind_configure VALUES ('organization','max entropy','en-ner-
```

```
organization.bin');
INSERT INTO nameFind_configure VALUES ('date','rules','date.rules');
INSERT INTO nameFind_configure VALUES ('time','rules','time.rules');
INSERT INTO nameFind_configure VALUES ('phone','rules','phone.rules');
INSERT INTO nameFind_configure VALUES ('money','rules','money.rules');
INSERT INTO nameFind_configure VALUES ('email','rules','email.rules');
INSERT INTO nameFind_configure VALUES ('percentage','rules','percentage.rules');
```

#### Default English-Language Models in Table nameFind\_configure

model_name	model_type	model_file
person	max entropy	en-ner-person.bin
location	max entropy	en-ner-location.bin
organization	max entropy	en-ner-organization.bin
date	rules	date.rules
time	rules	time.rules
phone	rules	phone.rules
money	rules	money.rules
email	rules	email.rules
percentage	rules	percentage.rules

## NamedEntityFinder Input

### Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Contains input text.
<i>accumulate_column</i>	Any	Column to copy to output table.

### ConfigurationTable Schema

This table is optional.

Column	Data Type	Description
<i>model_name</i>	VARCHAR	Name of an entity type (for example, PERSON, LOCATION, or EMAIL).
<i>model_type</i>	VARCHAR	One of these model types:

Column	Data Type	Description										
		<table><tr><th><i>model_type</i></th><th>Description</th></tr><tr><td>'max entropy'</td><td>Maximum entropy language model created by training</td></tr><tr><td>'rule'</td><td>Rule-based model, a plain text file with one regular expression on each line</td></tr><tr><td>'dictionary'</td><td>Dictionary-based model, a plain text file with one word on each line</td></tr><tr><td>'reg exp'</td><td>Regular expression that describes <i>entity_type</i></td></tr></table>	<i>model_type</i>	Description	'max entropy'	Maximum entropy language model created by training	'rule'	Rule-based model, a plain text file with one regular expression on each line	'dictionary'	Dictionary-based model, a plain text file with one word on each line	'reg exp'	Regular expression that describes <i>entity_type</i>
		<i>model_type</i>	Description									
		'max entropy'	Maximum entropy language model created by training									
		'rule'	Rule-based model, a plain text file with one regular expression on each line									
		'dictionary'	Dictionary-based model, a plain text file with one word on each line									
'reg exp'	Regular expression that describes <i>entity_type</i>											
<i>model_file</i>	VARCHAR	Name of model file that describes the entity type. This column appears if <i>model_type</i> is not 'reg exp'.										
reg_exp	VARCHAR	Regular expression that describes the entity type. This column appears if <i>model_type</i> is 'reg exp'.										

## NamedEntityFinder Output

### Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
entity_type	VARCHAR	Entity type.
entity	VARCHAR	Entity name.
entity_start	INTEGER	[Column appears only with ShowEntityContext syntax element.] Start position.
entity_end	INTEGER	[Column appears only with ShowEntityContext syntax element.] End position.
context	VARCHAR	[Column appears only with ShowEntityContext syntax element.] Words before and after the entity.

## NamedEntityFinder Example

### Input

Input Table: assortedtext\_input

id	source	content
1001	misc	contact Alan by email at sports@espn.com for all sport info

id	source	content
1002	misc	contact Mark at cricket@espn.com for all cricket info
1003	misc	contact Roger at tennis@espn.com for all tennis info
1004	wiki	The contiguous United States consists of the 48 adjoining U.S. states plus Washington, D.C., on the continent of North America
1005	wiki	California's economy is centered onTechnology,Finance,real estate services, Government, and professional, Scientific and Technical business Services; together comprising 58% of the State Government economy
1006	wiki	Houston is the largest city in Texas and the fourth-largest in the United States, while San Antonio is the second largest and seventh largest in the state.
1007	wiki	Thomas is a photographer whose natural landscapes of the West are also a statement about the importance of the preservation of the wildness

## SQL Call

```
SELECT * FROM NamedEntityFinder (
  ON assortedtext_input PARTITION BY ANY
  ON namefind_configure AS ConfigurationTable DIMENSION
  USING
    TextColumn ('content')
    Models ('all')
    Accumulate ('id', 'source')
) AS dt ORDER BY id;
```

## Output

id	source	entity	entity_type
----	-----	-----	-----
1001	misc	sports@espn.com	email
1002	misc	cricket@espn.com	email
1002	misc	Mark	person
1003	misc	Roger	person
1003	misc	tennis@espn.com	email
1004	wiki	Washington	location
1004	wiki	U.S.	location
1004	wiki	North America	location
1004	wiki	United States	location
1005	wiki	State Government	organization
1005	wiki	58%	percentage
1006	wiki	San Antonio	location
1006	wiki	United States	location

1006 wiki	Texas	location
1007 wiki	Thomas	person

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Named Entity Finder Evaluator

The `NamedEntityFinderEvaluatorMap` and `NamedEntityFinderEvaluatorReduce` functions operate as a row and a partition function, respectively. Each function takes a set of evaluating data and creates the precision, recall, and F-measure values of a specified maximum entropy data model. Neither function supports regular-expression-based or dictionary-based models.

### Related Information:

[Nondeterministic Results and UniqueID Syntax Element](#)

## Named Entity Finder Evaluator Syntax

### **NamedEntityFinderEvaluatorReduce version 1.5, NamedEntityFinderEvaluatorMap version 1.7**

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON { table | view | (query) }
    USING
    TextColumn ('text_column')
    InputModelFile ('input_model_file')
  ) AS alias_1 PARTITION BY 1
) AS alias_2;
```

## Named Entity Finder Evaluator Syntax Elements

### **TextColumn**

Specify the name of the input table column that contains the text to analyze.

### **InputModelFile**

Specify name of the model file to evaluate.

## NamedEntityFinderEvaluatorMap Input

### Input Table Schema

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax: <div>&lt;START:entity_type&gt; entity &lt;END&gt;</div> For example: <div>&lt;START:location&gt;Country1&lt;END&gt; has arrived</div>

## NamedEntityFinderEvaluatorReduce Output

### Output Table Schema

Column	Data Type	Description
precision_val	INTEGER	Precision value of the model.
recall	DOUBLE PRECISION	Recall value of the model.
f_measure	DOUBLE PRECISION	F-measure ( $F_1$ score) of the model.

## Named Entity Finder Evaluator Example

### Input

- Input Table: nermem\_sports\_test, which has rows of sports news
- model\_file*: location.sports, output by [NamedEntityFinderTrainer Example](#)

#### Input Table: nermem\_sports\_test

id	content
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>
9	<START:PER> Hussain <END>

id	content
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
14	Australian <START:PER> Tom Moody <END> took six for 82 but <START:PER> Chris Adams <END>
16	They were held up by a gritty 84 from <START:PER> Paul Johnson <END> but ex-England fast bowler <START:PER> Martin McCague <END> took four for 55 .
20	<START:LOCATION> LONDON <END> 1996-08-30
22	<START:LOCATION> Leicester <END> : <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs .
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON nermem_sports_test
    USING
    InputModelFile ('location.sports')
    TextColumn ('content')
  ) PARTITION BY 1
) AS dt;
```

## Output

```
precision_val    recall          f_measure
-----
0.847457627118644 0.7936507936507936 0.819672131147541
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

ACCESSING THIS CUSTOM DOCUMENT BEAR THE ENTIRE RISK OF ANY RELIANCE ON THIS CUSTOM DOCUMENT, INCLUDING AS TO QUALITY, ACCURACY, AND RESULTS.

## NGrams

The NGrams function *tokenizes* (splits) an input stream of text and outputs n multigrams (called n -grams) based on the specified delimiter and reset parameters. NGrams provides more flexibility than standard tokenization when performing text analysis. Many two-word phrases carry important meaning (for example, "machine learning") that unigrams (single-word tokens) do not capture. This, combined with additional analytical techniques, can be useful for performing sentiment analysis, topic identification, and document classification.

NGrams considers each input row to be one document, and returns a row for each unique n-gram in each document. NGrams also returns, for each document, the counts of each n-gram and the total number of n-grams.

For general information about tokenization, see [http://en.wikipedia.org/wiki/Lexical\\_analysis#Tokenizer](http://en.wikipedia.org/wiki/Lexical_analysis#Tokenizer).

## NGrams Syntax

### Version 1.8

```
SELECT * FROM NGrams (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')
  [ Delimiter ('delimiter_regular_expression') ]
  Grams ({ gram_number | 'value_range' }[,...])
  [ OverLapping ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ ToLowerCase ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ Reset ('reset_regular_expression') ]
  [ Punctuation ('punctuation_regular_expression') ]
  [ TotalGramCount ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ TotalCountColumn ('total_count_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ NGramColumn ('ngram_column') ]
  [ NumGramsColumn ('numgrams_column') ]
  [ FrequencyColumn ('count_column') ]
) AS alias;
```

Related information

- Column Specification Arguments
- Regular Expressions in Arguments

## NGrams Arguments

TextColumn

Specify the name of the column that contains the input text. This column must have a SQL string data type.

#### Delimiter

[Optional] Specify, with a regular expression, the character or string that separates words in the input text.

Default: "\\s+" (all whitespace characters—space, tab, newline, carriage return and others)

#### Grams

Specify the length, in words, of each n-gram (that is, the value of n). A value\_range has the syntax integer1-integer2, where integer1 <= integer2. The values of n, integer1, and integer2 must be positive.

#### OverLapping

[Optional] Specify whether the function allows overlapping n-grams.

Default: 'true' (Each word in each sentence starts an n-gram, if enough words follow it in the same sentence to form a whole n-gram of the specified size. For information on sentences, see the Reset argument description.)

#### ToLowerCase

[Optional] Specify whether the function converts all letters in the input text to lowercase.

Default: 'true'

#### Reset

[Optional] Specify, with a regular expression, the character or string that ends a sentence. At the end of a sentence, the function discards any partial n-grams and searches for the next n-gram at the beginning of the next sentence. An n-gram cannot span sentences.

The function applies the Reset argument before the Punctuation argument; that is, it splits the input into sentences before removing punctuation characters.

Default: '[.,?!]'

#### Punctuation

[Optional] Specify, with a regular expression, the punctuation characters for the function to remove before evaluating the input text.

The function applies the Reset argument before the Punctuation argument; that is, it splits the input into sentences before removing punctuation characters.

Default: '[`~#^&\*()-]'

#### TotalGramCount

[Optional] Specify whether the function returns the total number of n-grams in the document (that is, in the row) for each length n specified in the Grams argument. If you specify 'true', the TotalCountColumn argument determines the name of the output table column that contains these totals.

The total number of n-grams is not necessarily the number of unique n-grams.

Default: 'false'

#### TotalCountColumn

[Optional] Specify the name of the output table column that appears if the value of the TotalGramCount argument is 'true'.

Default: 'totalcnt'

#### Accumulate

[Optional] Specify the names of the input table columns to copy to the output table for each n-gram.

These columns cannot have the same names as those specified by the arguments NGramColumn, NumGramsColumn, and TotalCountColumn.

Default: All input columns for each n-gram

#### NGramColumn

[Optional] Specify the name of the output table column that is to contain the created n-grams.

Default: 'ngram'

NumGramsColumn

[Optional] Specify the name of the output table column that is to contain the length of n-gram (in words).

Default: 'n'

FrequencyColumn

[Optional] Specify the name of the output table column that is to contain the count of each unique n-gram (that is, the number of times that each unique n-gram appears in the document).

Default: 'frequency'

## NGrams Input

### Input Table Schema

Each row of the table has a document to tokenize. The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Document to tokenize.
accumulate_column	VARCHAR	[Column appears once for each specified accumulate_column.] Column to copy to output table.

## NGrams Output

### Output Table Schema

The table has a row for each unique n-gram in each input document.

Column	Data Type	Description
accumulate_column	VARCHAR	[Column appears once for each specified accumulate_column.] Column copied from input table.
ngram_column	VARCHAR	Created n-gram.
numgrams_column	INTEGER	Length of n-gram in words (value n).
count_column	INTEGER	Count of each unique n-gram in document.
total_count_column	INTEGER	[Column appears only with TotalCountColumn ('true').] Total number of n-grams in document for each length n specified in Grams argument.

## NGrams Examples

## NGrams Example 1: Overlapping ('true'), TotalGramCount ('true')

### Input

- Input Table: paragraphs\_input, which has paragraphs about common analytics topics (regression, decision Trees, and so on)

Input Table: paragraphs\_input

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of n points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible
...	...	...

## SQL Call

```
SELECT * FROM NGrams (
  ON paragraphs_input
  USING
  TextColumn ('paratext')
  Delimiter (' ')
  Grams ('4-6')
  OverLapping ('true')
  ToLowerCase ('true')
  Reset ('[.,?!]')
  Punctuation ('[`~#^&*()-]')
  TotalGramCount ('true')
  Accumulate ('paraid', 'paratopic')
) AS dt ORDER BY paraid, paratopic, ngram;
```

## Output

paraid	paratopic	ngram	n	frequency	totalcnt
1	Decision Trees	decision tree	4	1	73
1	Decision Trees	learning uses	5	1	66
1	Decision Trees	decision tree	6	1	60
1	Decision Trees	learning uses a	4	1	73
1	Decision Trees	tree learning	5	1	66
1	Decision Trees	uses a	6	1	60
1	Decision Trees	tree learning	4	1	73
1	Decision Trees	uses a decision	5	1	66
1	Decision Trees	tree learning	6	1	60
1	Decision Trees	learning uses a	4	1	73
1	Decision Trees	decision	5	1	66
1	Decision Trees	learning uses a	6	1	60
...	...	...	...	...	...

## NGrams Example 2: Overlapping ('false'), TotalGramCount ('false')

### Input

- Input Table: paragraphs\_input, as in NGrams Example 1: Overlapping ('true'), TotalGramCount ('true')

### SQL Call

```
SELECT * FROM NGrams (
  ON paragraphs_input
  USING
  TextColumn ('paratext')
  Delimiter (' ')
  Grams ('4-6')
  OverLapping ('false')
  ToLowerCase ('true')
  TotalGramCount ('false')
  Accumulate ('paraid', 'paratopic')
) AS dt ORDER BY paraid, paratopic, ngram;
```

### Output

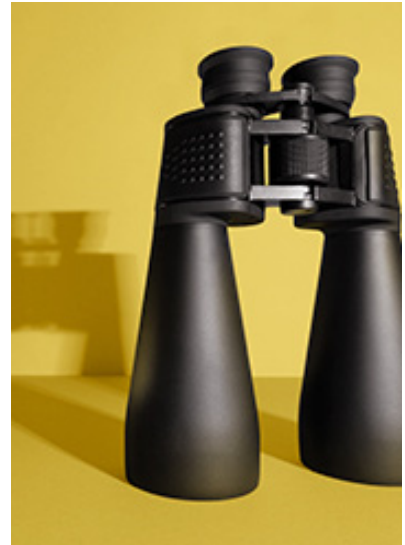
paraid	paratopic	ngram	n	frequency
1	Decision Trees	decision tree learning uses	4	1
1	Decision Trees	a decision tree as	4	1
1	Decision Trees	a predictive model which	4	1
1	Decision Trees	maps observations about an	4	1
1	Decision Trees	item to conclusions about	4	1
1	Decision Trees	the items target value	4	1
1	Decision Trees	decision tree learning uses a	5	1
1	Decision Trees	decision tree as a predictive	5	1
1	Decision Trees	model which maps observations about	5	1
1	Decision Trees	an item to conclusions about	5	1

paraid	paratopic	ngram	n	frequency
1	Decision Trees	decision tree learning uses a decision	6	1
1	Decision Trees	tree as a predictive model which	6	1
1	Decision Trees	maps observations about an item to	6	1
1	Decision Trees	conclusions about the items target value	6	1
...	...	...	...	...

## Current Topic – SentimentExtractor Background Information

teradata.

- NGrams
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - Review
- **SentimentExtractor**
  - **Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)**
  - Labs
  - Review



## SentimentExtractor Description (1 of 3)

- *Sentiment extraction* is the process of inferring user sentiment (*positive*, *negative*, or *neutral*) from text (typically call center logs, forums, and social media)
- The sentiment extraction functions support English, Simplified Chinese, and Traditional Chinese text
- There are three Sentiment Extraction functions. This document will cover only the **SentimentExtractor** function, employing a *dictionary model*
  - **SentimentTrainer**: Trains model. Takes training documents and outputs maximum entropy classification model
  - **SentimentExtractor**: Uses either classification model or dictionary model to extract sentiment of each input document or sentence; that is, to output predictions
  - **SentimentEvaluator**: Uses test data to evaluate precision and recall of predictions

Sentiment extraction is the process of inferring user sentiment (positive, negative, or neutral) from text (typically call center logs, forums, and social media).

This document will cover the **SentimentExtractor** function.

## SentimentExtractor Description (2 of 3)

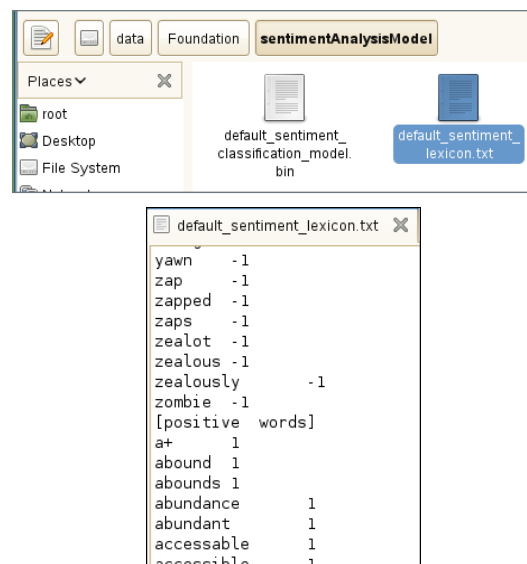
- The **SentimentExtractor** function extracts the sentiment (**positive**, **negative**, or **neutral**) of each input document or sentence, using either a classification model output by the function **SentimentTrainer** or a **dictionary model**
- The **dictionary model** consists of WordNet, a lexical database of the English language, and these negation words:
  - *no, not, neither, never, scarcely, hardly, nor, little, nothing, seldom, few*
- The function handles negated sentiments as follows:
  - **-1** if the sentiment is negated (for example, "*I am not happy*")
  - **-1** if the sentiment and a negation word are separated by one word (for example, "*I am not very happy*")
  - **+1** if the sentiment and a negation word are separated by two or more words (for example, "*I am not saying I am happy*")

The **SentimentExtractor** function extracts the sentiment (positive, negative, or neutral) of each input document or sentence, using either a classification model output by the function **SentimentTrainer** or a dictionary model.

This document will cover the dictionary model.

## SentimentExtractor Description (3 of 3)

- **Sentiment Extraction** is the process of deducing user opinion (positive, negative, neutral) from textual data
- Useful for analyzing people's opinions as found in forums, social media, and product reviews
- In this module, we will use a *Dictionary* approach, which scans through a dictionary file on the Machine Learning engine in an effort to determine the sentiment of selected text
- **Bottom line:** Decipher *feeling* behind users' words and phrases



- Sentiment Extraction is the process of deducing user opinion (positive, negative, neutral) from textual data
- Useful for analyzing people's opinions as found in forums, social media, and product reviews
- In this module, we will use a Dictionary approach, which scans through a dictionary file on the Machine Learning engine in an effort to determine the sentiment of selected text
- Bottom line: Decipher feeling behind users' words and phrases

## SentimentExtractor Use Cases

The **SentimentExtractor** function is all about scanning through text-based data in an attempt to discover the overall sentiment of the text.

This function could be used by any business in any industry.

Following are some examples:

- A **health care insurance** company wishes to scan through patient reviews of hospitals, clinics, and doctors regarding the quality of their care
- A **retailer** wishes to monitor online social media sites to discover user sentiment about the company, its products, etc.
- A **telecommunications company** wishes to scrutinize customer-support logs to discover which "call types" have a predominantly negative sentiment; i.e., *customer dissatisfaction*

The **SentimentExtractor** function could be useful for any business that wishes to deduce user sentiment based upon data in text form.

## SentimentExtractor Workflow



- **Input Tables:** Data is read from a specified input table, views, or query
- **SentimentExtractor:** There is only one required argument that must be specified when the function is invoked:
  - **TextColumn**
- **Console or Output table:** Data is written to the console or to an output table

The **SentimentExtractor** function will read from a defined table, view, or query, and output the results as defined by its arguments.

## SentimentExtractor Syntax

```
SELECT * FROM SentimentExtractor (
ON { table | view | (query) } [ PARTITION BY ANY ]
[ ON { table | view | (query) } AS dict DIMENSION ]
USING
TextColumn ('text_column')
[ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
[ ModelFile ({ 'dictionary[:dict_file]' | 'classification:model_file' }) ]
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
[ AnalysisType ({ 'document' | 'sentence' }) ]
[ Priority ({ 'NONE' |
'NEGATIVE_RECALL' |
'NEGATIVE_PRECISION' |
'POSITIVE_RECALL' |
'POSITIVE_PRECISION'}) ]
[ OutputType ({ 'ALL' | 'POSITIVE' | 'NEGATIVE' }) ]
) AS alias;
```

Here we are displaying the syntax structure for **SentimentExtractor**. Note that there is only one required argument.

**TextColumn**: Specify the name of the input column that contains text from which to extract sentiments

Note the following:

1. As with other Teradata Vantage functions, we are invoking the function through the call **SELECT \* FROM function\_name**; i.e., in this case, **SELECT \* FROM NGrams**.
2. Our input data can be in the form of a table, view, or query. It follows the **ON** keyword.

## SentimentExtractor Required Arguments

There is only one required argument for the **SentimentExtractor** function:

- **TextColumn**: Specify the name of the input column that contains text from which to extract sentiments

**TextColumn** is the only required input. It should contain the text from which we wish to deduce sentiment.

## SentimentExtractor Optional Arguments (1 of 4)

The following arguments available to you for the **SentimentExtraction** function are optional.

- **InputLanguage** [Optional]: Specify the language of the input text.
  - 'en' (Default) English
  - 'zh\_CN' Simplified Chinese
  - 'zh\_TW' Traditional Chinese
- **ModelFile** [Optional]: Specify the model type and file. The default model type is **dictionary**
  - If you omit this argument or specify **dictionary** without **dict\_file**, then you must specify a dictionary table with alias '**dict**'. If you specify both **dict** and **dict\_file**, then whenever their words conflict, **dict** has higher priority. The **dict\_file** must be a text file in which each line contains only a sentiment word, a space, and the opinion score of the sentiment word
  - If you specify classification **model\_file**, then **model\_file** must be the name of a model file created and installed on the ML Engine by the function **SentimentTrainer**

The following arguments available to you for the **SentimentExtraction** function are optional.

- **InputLanguage** [Optional]: Specify the language of the input text.
- **ModelFile** [Optional]: Specify the model type and file. The default model type is *dictionary*.
- **Accumulate** [Optional]: Specify the names of the input columns to copy to the output table
- **AnalysisType** [Optional]: Specify the level of analysis, whether to analyze each **document** (the default) or each **sentence**. A value of **document** refers to each row of input data, whereas a value of **sentence** refers to each sentence within each row of input data
- **Priority** [Optional]: Specify the highest priority when returning results. Following are the options available to you:
  - 'NONE' (Default) Give all results same priority
  - 'NEGATIVE\_RECALL' Give highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
  - 'NEGATIVE\_PRECISION' Give highest priority to negative results with high-confidence sentiment classifications
  - 'POSITIVE\_RECALL' Give highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned)
  - 'POSITIVE\_PRECISION' Give highest priority to positive
- **OutputType** [Optional]: Specify the kind of results to return. Following are the options available to you:
  - 'ALL' (Default) Return all results
  - 'POSITIVE' Return only results with positive sentiments
  - 'NEGATIVE' Return only results with negative sentiments

## SentimentExtractor Optional Arguments (2 of 4)

- **Accumulate** [Optional]: Specify the names of the input columns to copy to the output table
- **AnalysisType** [Optional]: Specify the level of analysis, whether to analyze each document (the default) or each sentence
  - A value of **document** refers to each row of input data, whereas a value of **sentence** refers to each sentence within each row of input data

The following arguments available to you for the **SentimentExtraction** function are optional.

- **InputLanguage** [Optional]: Specify the language of the input text.
- **ModelFile** [Optional]: Specify the model type and file. The default model type is *dictionary*.
- **Accumulate** [Optional]: Specify the names of the input columns to copy to the output table
- **AnalysisType** [Optional]: Specify the level of analysis, whether to analyze each **document** (the default) or each **sentence**. A value of **document** refers to each row of input data, whereas a value of **sentence** refers to each sentence within each row of input data
- **Priority** [Optional]: Specify the highest priority when returning results. Following are the options available to you:
  - 'NONE' (Default) Give all results same priority
  - 'NEGATIVE\_RECALL' Give highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
  - 'NEGATIVE\_PRECISION' Give highest priority to negative results with high-confidence sentiment classifications
  - 'POSITIVE\_RECALL' Give highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned)
  - 'POSITIVE\_PRECISION' Give highest priority to positive
- **OutputType** [Optional]: Specify the kind of results to return. Following are the options available to you:
  - 'ALL' (Default) Return all results
  - 'POSITIVE' Return only results with positive sentiments
  - 'NEGATIVE' Return only results with negative sentiments

## SentimentExtractor Optional Arguments (3 of 4)

- **Priority** [Optional]: Specify the highest priority when returning results. Following are the options available to you:
  - **'NONE'** (Default) Give all results same priority
  - **'NEGATIVE\_RECALL'** Give highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
  - **'NEGATIVE\_PRECISION'** Give highest priority to negative results with high-confidence sentiment classifications
  - **'POSITIVE\_RECALL'** Give highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned)
  - **'POSITIVE\_PRECISION'** Give highest priority to positive

The following arguments available to you for the **SentimentExtraction** function are optional.

- **InputLanguage** [Optional]: Specify the language of the input text.
- **ModelFile** [Optional]: Specify the model type and file. The default model type is *dictionary*.
- **Accumulate** [Optional]: Specify the names of the input columns to copy to the output table
- **AnalysisType** [Optional]: Specify the level of analysis, whether to analyze each **document** (the default) or each **sentence**. A value of **document** refers to each row of input data, whereas a value of **sentence** refers to each sentence within each row of input data
- **Priority** [Optional]: Specify the highest priority when returning results. Following are the options available to you:
  - **'NONE'** (Default) Give all results same priority
  - **'NEGATIVE\_RECALL'** Give highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
  - **'NEGATIVE\_PRECISION'** Give highest priority to negative results with high-confidence sentiment classifications
  - **'POSITIVE\_RECALL'** Give highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned)
  - **'POSITIVE\_PRECISION'** Give highest priority to positive
- **OutputType** [Optional]: Specify the kind of results to return. Following are the options available to you:
  - **'ALL'** (Default) Return all results
  - **'POSITIVE'** Return only results with positive sentiments
  - **'NEGATIVE'** Return only results with negative sentiments

## SentimentExtractor Optional Arguments (4 of 4)

- **OutputType** [Optional]: Specify the kind of results to return. Following are the options available to you:
  - **'ALL'** (Default) Return all results
  - **'POSITIVE'** Return only results with positive sentiments
  - **'NEGATIVE'** Return only results with negative sentiments

The following arguments available to you for the **SentimentExtraction** function are optional.

- **InputLanguage** [Optional]: Specify the language of the input text.
- **ModelFile** [Optional]: Specify the model type and file. The default model type is *dictionary*.
- **Accumulate** [Optional]: Specify the names of the input columns to copy to the output table
- **AnalysisType** [Optional]: Specify the level of analysis, whether to analyze each **document** (the default) or each **sentence**. A value of **document** refers to each row of input data, whereas a value of **sentence** refers to each sentence within each row of input data
- **Priority** [Optional]: Specify the highest priority when returning results. Following are the options available to you:
  - **'NONE'** (Default) Give all results same priority
  - **'NEGATIVE\_RECALL'** Give highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
  - **'NEGATIVE\_PRECISION'** Give highest priority to negative results with high-confidence sentiment classifications
  - **'POSITIVE\_RECALL'** Give highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned)
  - **'POSITIVE\_PRECISION'** Give highest priority to positive
- **OutputType** [Optional]: Specify the kind of results to return. Following are the options available to you:
  - **'ALL'** (Default) Return all results
  - **'POSITIVE'** Return only results with positive sentiments
  - **'NEGATIVE'** Return only results with negative sentiments

## SentimentExtractor Input Table Schema (required)

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text from which to extract sentiment
<i>accumulate_column</i>	VARCHAR	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table

Here is the input table schema.

## SentimentExtractor dict Table Schema (optional)

This table is optional. The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>sentiment_word</i>	VARCHAR	First column, containing the Sentiment word
<i>opinion_score</i>	INTEGER	Second column, containing opinion score for sentiment word

In our lab environment, we are reading dictionary contents from a file.

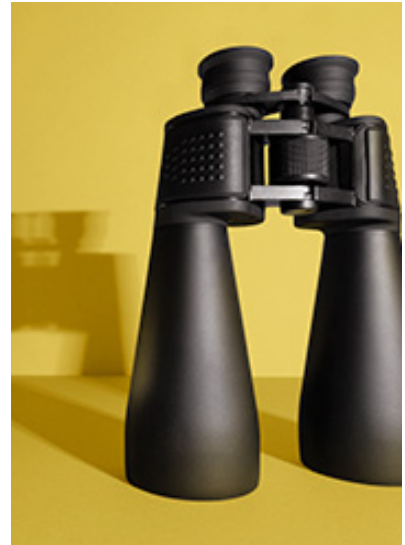
## SentimentExtractor Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as input table	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table
<i>out_content</i>	VARCHAR	[Column appears only for AnalysisType ('SENTENCE').] Displays the sentence that receives a sentiment score.
<i>out_polarity</i>	VARCHAR	Depends on value of out_content: <ul style="list-style-type: none"> <li>• If out_content NULL, then Nothing</li> <li>• If out_content Empty string, then UNKNOWN</li> <li>• If out_content any other value, then POS (positive), NEG (negative), or NEU (neutral)</li> </ul>
<i>out_strength</i>	INTEGER	Strength of output_polarity: <ul style="list-style-type: none"> <li>• 0: Neutral</li> <li>• 1: Higher than 0</li> <li>• 2: Higher than 1</li> </ul>
<i>out_sentiment_words</i>	VARCHAR	[Column appears only when function uses dictionary model.] Sentiment words in document or sentence

Here is the output table schema.

## Current Topic – SentimentExtractor Labs

- NGrams
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - Review
- **SentimentExtractor**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - **Labs**
  - Review





## Lab 06a – Fundamentals of SentimentExtractor

```
SELECT *  
FROM bb_sentiment0  
ORDER BY x_text;
```

- Here, we are familiarizing ourselves with the source input table against which we will run the **SentimentExtractor** function
- The next few pages will illustrate how to run a **SentimentExtractor** query against this input table—using the **TextColumn** (required) and **Accumulate** (optional) arguments
- Examples in this document will utilize the *Dictionary* format for extracting sentiment

Input Data

	x_text
1	bad
2	bad bad good
3	bad good
4	good
5	good bad
6	good good bad
7	great
8	I am not happy
9	I am not saying I am happy
10	I am not saying I am not happy
11	I am not saying I am very happy
12	I am not very happy
13	neutral
14	pretty
15	terrible
16	ugly

In this lab, we are familiarizing ourselves with the basic syntax and output of **SentimentExtractor**.



## Lab 06b – Fundamentals of SentimentExtractor

```
SELECT *  
FROM SentimentExtractor (  
ON bb_sentiment0  
USING  
TextColumn ('x_text')  
Accumulate ('x_text')  
) AS dt  
ORDER BY x_text;
```

The **SentimentExtractor** function is structured like other SQL Engine queries.

```
SELECT * FROM function_name  
ON my_table  
USING  
argument_a,  
argument_b ...
```

The query above is using the sole required argument and one optional argument:

- **TextColumn** (required): Here, we are specifying the name of the column that contains the data on which we wish to discover *sentiment*. By default, each row of data will receive its own sentiment score
- **Accumulate** (optional): Here, we are specifying that we wish to include the **x\_text** column in the output

In this lab, we are familiarizing ourselves with the basic syntax and output of **SentimentExtractor**.



## Lab 06c – Fundamentals of SentimentExtractor

- Our output contains by default the three columns listed below:
  - out\_polarity** displays the sentiment. **POS** (positive), **NEU** (neutral), **NEG** (negative)
  - out\_strength** displays the strength of **out\_polarity**. Possible values are **0**, **1**, and **2** (with **2** being the strongest)
  - out\_sentiment\_words** displays each word from the input text that was found in the dictionary and scores it accordingly (**+1** for positive words and **-1** for negative words)
- The **x\_text** column was returned as a result of our optional **Accumulate** argument

### Partial Answer Set

	x_text	out_polarity	out_strength	out_sentiment_words
1	bad	NEG	2	bad -1. In total, positive score:0 negative score:-1
2	bad bad good	NEG	2	bad -1, bad -1, good 1. In total, positive score:1 negative score:-2
3	bad good	NEU	0	bad -1, good 1. In total, positive score:1 negative score:-1
4	good	POS	2	good 1. In total, positive score:1 negative score:0
5	good bad	NEU	0	good 1, bad -1. In total, positive score:1 negative score:-1
6	good good bad	POS	2	good 1, good 1, bad -1. In total, positive score:2 negative score:-1

In this lab, we are familiarizing ourselves with the basic syntax and output of **SentimentExtractor**.



## Lab 06d – Fundamentals of SentimentExtractor

	x text	out.polarity	out.strength
1	bad	NEG	2
2	bad bad good	NEG	2
3	I am not happy	NEG	2
4	I am not saying I am not happy	NEG	2
5	I am not very happy	NEG	2
6	terrible	NEG	2
7	ugly	NEG	2
8	bad good	NEU	0
9	good bad	NEU	0
10	neutral	NEU	0
11	good	POS	2
12	good good bad	POS	2
13	great	POS	2
14	I am not saying I am happy	POS	2
15	I am not saying I am very happy	POS	2
16	pretty	POS	2

- **NEG:** In each case, the negative terms are greater in number than the positive terms

- **NEU:** In each case, the term is either neutral itself, or the terms cancel one another out

- **POS:** In each case, the positive terms are greater in number than the negative terms. Also, note the attempt to deduce sentiment based upon negation words

In this lab, we are familiarizing ourselves with the basic syntax and output of **SentimentExtractor**.



## Lab 07a – OutputType All (Optional)

```
SELECT * FROM SentimentExtractor (  
ON bb_sentiment1  
USING  
TextColumn ('x_text')  
OutputType ('ALL')  
Accumulate ('x_text')  
) AS dt  
ORDER BY out_polarity, x_text;
```

- Here, we have explicitly injected the optional argument of **OutputType ('ALL')**. If we had left the argument out altogether, the query would have behaved the same exact way—as this is the default
- A value of **'ALL'** causes all rows to be included in the output, regardless of their **out\_polarity**

Input Data

	x_text
1	bad
2	bad bad
3	bad bad good
4	bad good
5	good
6	good bad
7	good good
8	good good bad
9	neutral

Output

	x_text	out_polarity	out_strength	out_sentiment_words
1	bad	NEG	2	bad -1. In total, positive score:0 negative score:-1
2	bad bad	NEG	2	bad -1, bad -1. In total, positive score:0 negative score:-2
3	bad bad good	NEG	2	bad -1, bad -1, good 1. In total, positive score:1 negative score:-2
4	bad good	NEU	0	bad -1, good 1. In total, positive score:1 negative score:-1
5	good bad	NEU	0	good 1, bad -1. In total, positive score:1 negative score:-1
6	neutral	NEU	0	
7	good	POS	2	good 1. In total, positive score:1 negative score:0
8	good good	POS	2	good 1, good 1. In total, positive score:2 negative score:0
9	good good bad	POS	2	good 1, good 1, bad -1. In total, positive score:2 negative score:-1

**OutputType** of **ALL** will return all three possible **out\_polarity** values. This is the default.



## Lab 07b – OutputType POSITIVE (Optional)

```
SELECT * FROM SentimentExtractor (  
ON bb_sentiment1  
USING  
TextColumn ('x_text')  
OutputType ('POSITIVE')  
Accumulate ('x_text')  
) AS dt  
ORDER BY out_polarity, x_text;
```

- Here, we have explicitly injected the optional argument of **OutputType** ('POSITIVE')
- A value of 'POSITIVE' causes only rows with an **out\_polarity** of **POS** to be included in the output

	x_text	out_polarity	out_strength	out_sentiment_words
1	good	POS	2	good 1. In total, positive score:1 negative score:0
2	good good	POS	2	good 1, good 1. In total, positive score:2 negative score:0
3	good good bad	POS	2	good 1, good 1, bad -1. In total, positive score:2 negative score:-1

**OutputType** of **POSITIVE** will return only **out\_polarity** values of **POS**.



## Lab 07c – OutputType NEGATIVE (Optional)

```
SELECT * FROM SentimentExtractor (  
ON bb_sentiment1  
USING  
TextColumn ('x_text')  
OutputType ('NEGATIVE')  
Accumulate ('x_text')  
) AS dt  
ORDER BY out_polarity, x_text;
```

- Here, we have explicitly injected the optional argument of **OutputType ('NEGATIVE')**
- A value of **'NEGATIVE'** causes only rows with an **out\_polarity** of **NEG** to be included in the output

	x_text	out_polarity	out_strength	out_sentiment_words
1	bad	NEG	2	bad -1. In total, positive score:0 negative score:-1
2	bad bad	NEG	2	bad -1, bad -1. In total, positive score:0 negative score:-2
3	bad bad good	NEG	2	bad -1, bad -1, good 1. In total, positive score:1 negative score:-2

**OutputType** of **NEGATIVE** will return only **out\_polarity** values of **NEG**.



## Lab 08a – AnalysisType Document

```
SELECT *  
FROM SentimentExtractor (  
ON bb_sentiment2  
USING  
TextColumn ('x_text')  
AnalysisType ('document')  
Accumulate ('x_text')  
) AS dt  
ORDER BY out_polarity, x_text;
```

- Here, we have explicitly injected the optional argument of **OutputType ('document')**. If we had left the argument out altogether, the query would have behaved the same exact way—as this is the default
- A value of **'document'** causes each row of data to receive a single score, regardless of how many sentences are within the row

Input Data

	x_text
1	I am happy. I am sad. It was a great day.
2	I am happy. I am sad. It was a terrible day.
3	I am happy. It was a great day.
4	I am sad. It was a terrible day.

Output (not showing out\_sentiment\_words column)

	x_text	out_polarity	out_strength
1	I am happy. I am sad. It was a terrible day.	NEG	2
2	I am sad. It was a terrible day.	NEG	2
3	I am happy. I am sad. It was a great day.	POS	2
4	I am happy. It was a great day.	POS	2

When we analyze at the **document** level, notice that *all sentences* within the row are evaluated as a collective. The entire row receives a single sentiment score.



## Lab 08b – AnalysisType Sentence

```
SELECT *  
FROM SentimentExtractor (  
ON bb_sentiment2  
USING  
TextColumn ('x_text')  
AnalysisType ('sentence')  
Accumulate ('x_text')  
) AS dt  
ORDER BY x_text,  
out_polarity, out_content;
```

### Input Data

	x_text
1	I am happy. I am sad. It was a great day.
2	I am happy. I am sad. It was a terrible day.
3	I am happy. It was a great day.
4	I am sad. It was a terrible day.

- Here, we have explicitly injected the optional argument of **OutputType ('sentence')**
- A value of **'sentence'** causes each sentence within each row of data to receive a single score
- Note the auto-creation of the **out\_content** column

### Output (not showing out\_sentiment\_words column)

	x_text	out_content	out_polarity	out_strength
1	I am happy. I am sad. It was a great day.	I am sad.	NEG	2
2	I am happy. I am sad. It was a great day.	I am happy.	POS	2
3	I am happy. I am sad. It was a great day.	It was a great day.	POS	2
4	I am happy. I am sad. It was a terrible day.	I am sad.	NEG	2
5	I am happy. I am sad. It was a terrible day.	It was a terrible day.	NEG	2
6	I am happy. I am sad. It was a terrible day.	I am happy.	POS	2
7	I am happy. It was a great day.	I am happy.	POS	2
8	I am happy. It was a great day.	It was a great day.	POS	2
9	I am sad. It was a terrible day.	I am sad.	NEG	2
10	I am sad. It was a terrible day.	It was a terrible day.	NEG	2

When we analyze at the **sentence** level, notice that *each sentence* within each row receives its own sentiment score, independent of any other sentences that may exist within the same row.



## Lab 09a – Priority (Source Data)

```
SELECT * FROM bb_sentiment ORDER BY x_id;
```

### Input Data

	x_id	x_text
1	0	good good good good good good good good good good
2	1	good good good good good good good good good bad
3	2	good good good good good good good good bad bad
4	3	good good good good good good good bad bad bad
5	4	good good good good good good bad bad bad bad
6	5	good good good good good bad bad bad bad bad
7	6	good good good good bad bad bad bad bad bad
8	7	good good good bad bad bad bad bad bad bad
9	8	good good bad bad bad bad bad bad bad bad
10	9	good bad bad bad bad bad bad bad bad bad
11	10	bad bad bad bad bad bad bad bad bad bad

Over the next many pages, we will leverage different values in the optional **Priority** argument to see how this impact the output of **SentimentExtractor**. Values include:

- None
- Negative Recall
- Negative Precision
- Positive Recall
- Positive Precision

- Our input data has eleven rows
- We start off with ten instances of the word **good** in **x\_id 0**, and each subsequent **x\_id** value replaces one of the words **good** with the word **bad**
- By the time we reach **x\_id 10**, all ten instances of **good** have been replaced with **bad**

This lab uses a simple data-set to discuss the various **Priority** values:

- **None**
- **Negative Recall**
- **Negative Precision**
- **Positive Recall**
- **Positive Precision**



## Lab 09b – Priority (None)

```
SELECT *  
FROM SentimentExtractor (  
ON bb_sentiment  
USING  
TextColumn ('x_text')  
Accumulate ('x_id')  
Priority ('NONE')  
) AS dt  
ORDER BY x_id;
```

x_id	good	bad	x_id	out_polarity	out_strength
0	10	0	0	POS	2
1	9	1	1	POS	2
2	8	2	2	POS	2
3	7	3	3	POS	2
4	6	4	4	POS	1
5	5	5	5	NEU	0
6	4	6	6	NEG	1
7	3	7	7	NEG	2
8	2	8	8	NEG	2
9	1	9	9	NEG	2
10	0	10	10	NEG	2

- A **Priority** of **None** gives all results the same priority
- In total, we have five **POS**, one **NEU**, and five **NEG**
  - If "good" > "bad", then **POS**
  - If "good" = "bad", then **NEU**
  - If "good" < "bad", then **NEG**

If not specified, the default **Priority** is **None**.



## Lab 09c – Priority (Negative\_Recall)

```
SELECT *  
FROM SentimentExtractor (  
ON bb_sentiment  
USING  
TextColumn ('x_text')  
Accumulate ('x_id')  
Priority ('NEGATIVE_RECALL')  
) AS dt  
ORDER BY x_id;
```

x_id	good	bad	x_id	out polarity	out strength
0	10	0	0	POS	2
1	9	1	1	POS	2
2	8	2	2	POS	2
3	7	3	3	POS	1
4	6	4	4	NEU	0
5	5	5	5	NEG	1
6	4	6	6	NEG	1
7	3	7	7	NEG	2
8	2	8	8	NEG	2
9	1	9	9	NEG	2
10	0	10	10	NEG	2

- A **Priority** of **Negative Recall** gives highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned)
- In total, we have four **POS**, one **NEU**, and six **NEG**
  - Note that an equal number of "good" and "bad" evaluates to **NEG**
  - Our value of six "good" and four "bad" evaluates to **NEU**
  - When it is a toss-up, **NEG** wins. When "good" slightly outnumbers "bad", **NEU** wins

**Negative\_Recall** maximizes the number of negative results.



## Lab 09d – Priority (Negative\_Precision) (Optional)

teradata.

```
SELECT * FROM SentimentExtractor (  
ON bb_sentiment  
USING  
TextColumn ('x_text')  
Accumulate ('x_id')  
Priority('NEGATIVE_PRECISION')  
) AS dt  
ORDER BY x_id;
```

x_id	good	bad	x_id	out_polarity	out_strength
0	10	0	0	POS	2
1	9	1	1	POS	2
2	8	2	2	POS	2
3	7	3	3	POS	2
4	6	4	4	POS	1
5	5	5	5	NEU	0
6	4	6	6	NEU	0
7	3	7	7	NEG	2
8	2	8	8	NEG	2
9	1	9	9	NEG	2
10	0	10	10	NEG	2

- A **Priority** of **Negative\_Precision** gives highest priority to negative results with high-confidence sentiment classifications
- In total, we have five **POS**, two **NEU**, and four **NEG**
  - Note that our value of four "good" and six "bad" evaluates to **NEU**
  - Only rows which are *predominantly "bad"* evaluate to **NEG**

**Negative\_Precision** attempts to only flag rows as negative if they are *predominantly negative*.



## Lab 09e – Priority (Positive\_Recall)

```
SELECT * FROM
SentimentExtractor (
ON bb_sentiment
USING
TextColumn ('x_text')
Accumulate ('x_id')
Priority ('POSITIVE_RECALL')
) AS dt
ORDER BY x_id;
```

x_id	good	bad	x_id	out polarity	out strength
0	10	0	0	POS	2
1	9	1	1	POS	2
2	8	2	2	POS	2
3	7	3	3	POS	2
4	6	4	4	POS	1
5	5	5	5	POS	1
6	4	6	6	NEU	0
7	3	7	7	NEG	1
8	2	8	8	NEG	2
9	1	9	9	NEG	2
10	0	10	10	NEG	2

- A **Priority** of **Positive\_Recall** gives highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned).
- In total, we have six POS, one **NEU**, and four **NEG**
  - Note that an equal number of "good" and "bad" evaluates to **POS**
  - Our value of four "good" and six "bad" evaluates to **NEU**
  - When it is a toss-up, **POS** wins. When "bad" slightly outnumbers "good", **NEU** wins

**Positive\_Recall** maximizes the number of positive results.



## Lab 09f – Priority (Positive\_Precision) (Optional)

```
SELECT * FROM
SentimentExtractor (
ON bb_sentiment
USING
TextColumn ('x_text')
Accumulate ('x_id')
Priority('POSITIVE_PRECISION')
) AS dt
ORDER BY x_id;
```

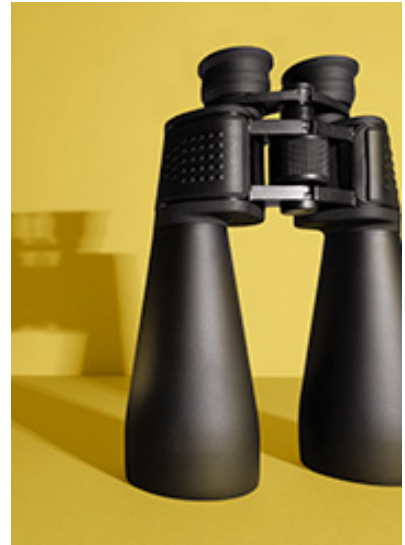
x_id	good	bad	x_id	out_polarity	out_strength
0	10	0	0	POS	2
1	9	1	1	POS	2
2	8	2	2	POS	2
3	7	3	3	POS	2
4	6	4	4	NEU	0
5	5	5	5	NEU	0
6	4	6	6	NEG	1
7	3	7	7	NEG	2
8	2	8	8	NEG	2
9	1	9	9	NEG	2
10	0	10	10	NEG	2

- A **Priority** of **Positive\_Precision** gives highest priority to positive results with high-confidence sentiment classifications
- In total, we have four POS, two **NEU**, and five **NEG**
  - Note that our value of six "good" and four "bad" evaluates to **NEU**
  - Only rows which are *predominantly "good"* evaluate to **POS**

**Positive\_Precision** attempts to only flag rows as negative if they are *predominantly positive*.

## Current Topic – SentimentExtractor Review

- NGrams
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - Review
- **SentimentExtractor**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - **Review**





## Hackathon: Product Reviews Sentiment

The following exercise is intended to provide you with further practice on using the **SentimentExtractor** function. There is no single "right" or "wrong" answer. The intent is for you to become comfortable writing queries that use **SentimentExtractor**

1. Run a **SentimentExtractor** query on the **bb\_sentiment\_extract\_input** table, which shows 10 reviews about various products. **What is the general sentiment expressed in each review?**
2. Things to think about follow:
  - What is a suitable Priority to specify?
  - After reading through the product reviews yourself, do you think that **SentimentExtractor** did a decent job of deducing the general sentiment of each product review?

In this "free-form" exercise, there are no "right" or "wrong" answers. The intent is to get you to write your own **SENTIMENT\_EXTRACTOR** query(ies) so as to become more comfortable with the syntax. Following are the contents of our source input data.

id	product	review
1	camera	we primarily bought this camera for high image quality and excellent video capability without paying the price for a dsir. it has excelled in what we expected of it, and consequently represented excellent value for me. all my friends want my camera for their vacations. i would recommend this camera to anybody. definitely worth the price. plus, when you buy some accessories, it becomes even more powerful.
2	office suite	it is the best office suite i have used to date. it is launched before office 2010 and it is ages ahead of it already. the fact that i could comfortably import xls, doc, ppt and modify them, and then export them back to the doc, xls, ppt is terrific. i needed the compatibility. it is a very intuitive suite and the drag drop functionality is terrific.
3	camera	this is a nice camera, delivering good quality video images decent photos. light small, using easily obtainable, high quality minidv i love it. minor irritations include touchscreen based menu only digital photos can only be transferred via usb, requiring ilink and usb if you use ilink.
4	gps	it is a fine gps. outstanding performance, works great. you can even get incredible coordinate accuracy from streets and trips to compare.
5	gps	nice graphs and map route info. i would not run outside again without this unique gadget. great job. big display, good backlight, really watertight, training assistant. i use in trail running and it worked well through out the race.
6	gps	most of the complaints i have seen in here are from a lack of rtfm. i have never seen so many mistakes do to what i think has to be none update of data to the system. i wish i could make all the rating stars be empty.
7	gps	this machine is all screwed up. on my way home from a friends house it told me there is no possible route. i found their website support difficult to navigate. i am is so disappointed and just returned it and now looking for another one
8	camera	i hate my camera, and im stuck with it. this camera sucks so bad, even the dealers on ebay have difficulty selling it. horrible indoors, does not capture fast action, screwy software, no suprise, and screwy audio/video codec that does not work with hardly any app.
9	television	\$3k is way too much money to drop onto a piece of crap. poor customer support. after about 1 and a half years and hardly using the tv, a big yellow pixilated stain appeared. product is very inferior and subject to several lawsuits. i expressed my dissatisfaction with the situation as this is a known issue.
10	camera	i returned my camera to the vendor as i will not tolerate a substandard product that is a known issue especially from vendor who will not admit that this needs to be removed from the shelf due to failing parts updated. due to the constant need for repair, i would never recommend this product.



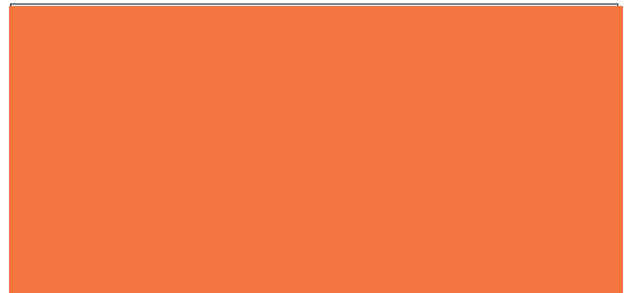
## Hackathon: Product Reviews Sentiment (Possible Answer)

teradata.

```
SELECT *  
FROM bb_sentiment_extract_input;
```

### Input Data

	id	product	review
1	1	camera	we primarily bought this camera for hi...
2	2	office suite	it is the best office suite i have used to...
3	3	camera	this is a nice camera, delivering good ...
4	4	gps	it is a fine gps. outstanding performan...
5	5	gps	nice graphs and map route info. i woul...
6	6	gps	most of the complaints i have seen in ...
7	7	gps	this machine is all screwed up. on my ...
8	8	camera	i hate my camera, and im stuck with it...
9	9	television	\$3k is way too much money to drop o...
10	10	camera	i returned my camera to the vendor as...



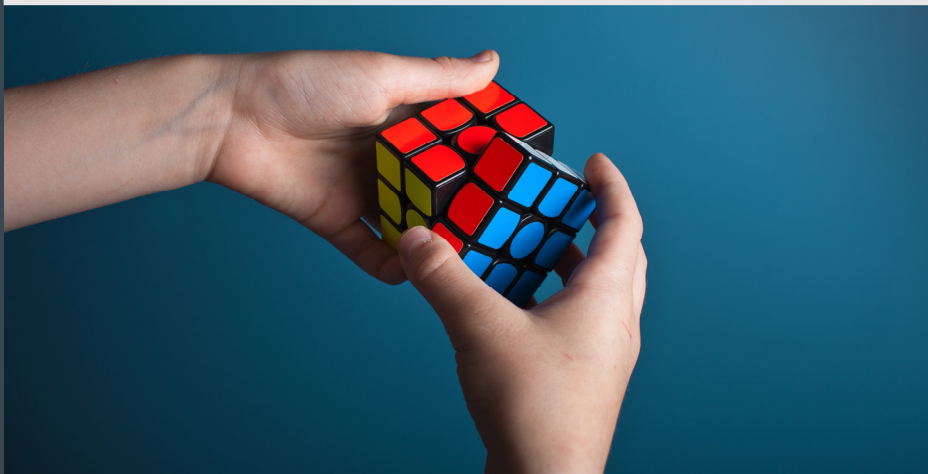
### Possible Answer-Set

	id	product	review	out polarity	out strength	out sentime...
1	8	camera	i hate my ca...	NEG	2	hate -1, stuck...
2	10	camera	i returned my...	NEG	2	substandard ...
3	1	camera	we primarily ...	POS	2	excellent 1, c...
4	3	camera	this is a nice ...	POS	2	nice 1, good ...
5	6	gps	most of the c...	NEG	2	complaints -...
6	7	gps	this machine ...	NEG	2	screwed -1, s...
7	4	gps	it is a fine gp...	POS	2	fine 1, outsta...
8	5	gps	nice graphs a...	POS	2	nice 1, great ...
9	2	office suite	it is the best ...	POS	2	best 1, comf...
10	9	television	\$3k is way to...	NEG	2	crap -1, poor...

In this “free-form” exercise, there are no “right” or “wrong” answers. The intent is to get you to write your own **SENTIMENT\_EXTRACTOR** query(ies) so as to become more comfortable with the syntax.

## Game Time! SentimentExtractor Million \$ Question

[Click here to start!](#)



This game, containing review questions, reinforces the module objectives for SentimentExtractor.

## Summary

In this module, you learned how to:

- Describe what the **NGrams** and **SentimentExtractor** functions do
- Describe typical use cases for **NGrams** and **SentimentExtractor**
- Write **NGrams** and **SentimentExtractor** queries
- Interpret the output of **NGrams** and **SentimentExtractor** queries

# Named Entity Recognition (NER) Functions (ML Engine)

*Named entity recognition (NER)* is a process for finding specified entities in text. For example, a simple news named-entity recognizer for English might find the person "John J. Smith" and the location "Seattle" in the text string "John J. Smith lives in Seattle."

NER functions let you specify how to extract named entities when training the data models. ML Engine provides two sets of NER functions:

Function Set	Supported Languages
<a href="#">NER Functions (CRF Model Implementation)</a>	English, simplified Chinese, traditional Chinese
<a href="#">NER Functions (Maximum Entropy Model Implementation)</a>	English

## NER Functions (CRF Model Implementation)

Function	Description
<a href="#">NERTrainer</a>	Takes training data and outputs CRF model (binary file).
<a href="#">NERExtractor</a>	Takes input documents and extracts specified entities, using one or more CRF models and, if appropriate, rules (regular expressions) or a dictionary. Uses models to extract names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and dictionary to extract known entities.
<a href="#">NEREvaluator</a>	Evaluates CRF model.

The CRF model implementation supports English, simplified Chinese, and traditional Chinese text.

### Related Information:

[NER Functions \(Maximum Entropy Model Implementation\)](#)

## NERTrainer

The NERTrainer function takes training data and outputs a CRF model (a binary file) that can be specified in the function [NERExtractor](#) and [NEREvaluator](#).

NERTrainer uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

### NERTrainer Syntax

**Version 1.8**

```

SELECT * FROM NERTrainer (
  ON { table | view | (query) } PARTITION BY 1
  USING
  ModelFileName (model_file)
  TextColumn ('text_column')
  [ ExtractorJAR ('jar_file') ]
  FeatureTemplate ('template_file')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ MaxIterNum (max_iteration_times) ]
  [ Eta (eta_threshold_value) ]
  [ MinOccurNum (threshold_value) ]
) AS alias;

```

**NERTrainer Syntax Elements****ModelFileName**

Specify the name of the model file that the function creates and installs on ML Engine.

**TextColumn**

Specify the name of the input table column that contains the text to analyze.

**ExtractorJAR**

[Optional] Specify the name of the JAR file that contains the Java classes that extract features. You must install this JAR file on ML Engine before calling the function.

The name *jar\_file* is case-sensitive.

ML Engine does not support the creation of new extractor classes. However, it does support existing JAR files—for installation instructions, see *Teradata Vantage™ User Guide*, B700-4002.

Default behavior: The function uses only the predefined extractor classes.

**FeatureTemplate**

Specify the name of the file that specifies how to create features when training the model.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**MaxIterNum**

[Optional] Specify the maximum number of iterations.

Default: 1000

**Eta**

[Optional] Specify the tolerance of the termination criterion. Defines the differences of the values of the loss function between two sequential epochs.

When training a model, the function performs  $n$ -times iterations. At the end of each epoch, the function calculates the loss or cost function on the training samples. If the loss function value change is very small between two sequential epochs, the function considers the training process to have converged.

The function defines Eta as:

$$\text{Eta} = (f(n) - f(n-1)) / f(n-1)$$

where  $f(n)$  is the loss function value of the  $n$ th epoch.

Default: 0.0001

**MinOccurNum**

[Optional] Specify the minimum number times that a feature must occur in the input text before the function uses the feature to construct the model.

Default: 0

**NERTrainer Input****Input Table Schema**

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
text_column	VARCHAR	Text to analyze. Within text, each entity must be identified with this syntax: <code>&lt;START:entity_type&gt;entity&lt;END&gt;</code> For example: <code>&lt;START:location&gt;Country1&lt;END&gt; has arrived</code>

**NERTrainer Output**

The function outputs a message and a CRF model (a binary file installed on ML Engine).

## Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Reports training time and file size of model.

## NERTrainer Example

### Input

- Input table: ner\_sports\_train, a collection of sports news items (500 rows)
- Feature template file: template\_1.txt, which is preinstalled on ML Engine.

#### ner\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOC> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOC> Grace Road <END>
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
...	...

### SQL Call

```
SELECT * FROM NERTrainer (
  ON ner_sports_train PARTITION BY 1
  USING
    TextColumn ('content')
    FeatureTemplate ('template_1.txt')
    OutputModelFile ('ner_model.bin')
) AS dt;
```

## Output

```
train_result
-----
Model generated.
Training time(s): 3.129
File size(KB): 374
Model successfully installed.
```

The model file, ner\_model.bin, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NERExtractor

The NERExtractor function takes input documents and extracts specified entities, using one or more CRF models (output by the function [NERTrainer](#)) and, if appropriate, rules (regular expressions) or a dictionary.

The function uses models to extract the names of persons, locations, and organizations; rules to extract entities that conform to rules (such as phone numbers, times, and dates); and a dictionary to extract known entities.

NERExtractor uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NERExtractor Syntax

### Version 1.8

```
SELECT * FROM NERExtractor (
  ON input_table PARTITION BY { ANY | key }
  [ ON rules_table AS Rules DIMENSION ]
  [ ON dictionary_table AS Dict DIMENSION ]
  USING
  TextColumn ('text_column')
  [ InputModelFiles ('input_model_file[:jar_file]' [,...]) ]
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
  [ ShowContext ('n') ]
```

```
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

**Related Information:**

[Column Specification Syntax Elements](#)

**NERExtractor Syntax Elements****TextColumn**

Specify the name of the input table column that contains the text to analyze.

**InputModelFiles**

[Optional] Specify the CRF models (binary files) to use, output by [NERTrainer](#). If you specified the ExtractorJAR syntax element in the NERTrainer call that created *input\_model\_file*, then you must specify the same *jar\_file* in this syntax element. You must install *input\_model\_file* and *jar\_file* in ML Engine before calling the NERExtractor function.

The names *input\_model\_file* and *jar\_file* are case-sensitive.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**ShowContext**

[Optional] Specify the number of context words to output (a positive integer). The function outputs the *n* words that precede the entity, the entity, and the *n* words that follow the entity.

Default: 0

**Accumulate**

[Optional] Specify the names of the input table columns to copy to the output table.

**NERExtractor Input**

Table	Description
Input table	Text to analyze. <b>Tip:</b> To optimize function performance, remove punctuation marks from text with <a href="#">TextParser (ML Engine)</a> function.
Rules	[Optional] Rules to use when extracting entities from text.

Table	Description
Dict	[Optional] Dictionary to use when extracting entities from text.

### Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze.
<i>accumulate_column</i>	Any	Column to copy to output table.

### Rules Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
regex	VARCHAR	Regular expression that represents an entity of this type. Expression must conform to Java Regex standard, documented at <a href="http://docs.oracle.com/javase/tutorial/essential/regex/quant.html">http://docs.oracle.com/javase/tutorial/essential/regex/quant.html</a> .

### Dict Schema

Column	Data Type	Description
type	VARCHAR	Entity type.
dict	VARCHAR	Dictionary word.

## NERExtractor Output

### Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
sn	INTEGER	Serial number of extracted entity.
entity	VARCHAR	Extracted entity.
type	VARCHAR	Type of extracted entity.
start	INTEGER	Start position of extracted entity in input text.
end	INTEGER	End position of extracted entity in input text.

Column	Data Type	Description
context	VARCHAR	[Column appears only with ShowContent syntax element.] Context of extracted entity.
approach	VARCHAR	Method used to identify extracted entity—CRF, RULE, or DICT.

## NERExtractor Example

### Input

- Input table: ner\_sports\_test2, which contains text to analyze.
- Rules: rule\_table, which is preinstalled on ML Engine.
- Model: ner\_model.bin, output by [NERTrainer Example](#).

#### Input table: ner\_sports\_test2

id	content
528	email sports@espn.com to contact for all sport info
529	email cricket@espn.com to contact for all cricket info
530	email tennis@espn.com to contact for all tennis info
531	1= Igor Trandenkov (Russia) 5.86
532	3. Maksim Tarasov (Russia) 5.86
533	4. Tim Lobinger (Germany) 5.80
534	5. Igor Potapovich (Kazakstan) 5.80
535	6. Jean Galfione (France) 5.65
536	7. Pyotr Bochkary (Russia) 5.65
537	8. Dmitri Markov (Belarus) 5.65
583	GENEVA 1996-08-30
584	UEFA came down heavily on Belgian club Standard Liege on Friday for disgraceful behaviour in an Intertoto final match against Karlsruhe of Germany .
585	The Belgian club were fined 25
586	He was sent off for insulting the referee and then urged his team mates to protest .
587	Roberto Bisconti will be sidelined for six Euro ties after pushing the referee in the back as he protested about a Karlsruhe goal
588	Karlsruhe won the August 20 match 3-1 thanks to two late goals .

id	content
589	They took the tie 3-2 on aggregate and qualified for the UEFA Cup .
591	ATHLETICS - HARRISON
592	MONTE CARLO 1996-08-30
593	Olympic champion Kenny Harrison and world record holder Jonathan Edwards will both take part in a triple jump competition at the Solidarity Meeting for Sarajevo on September 9 .
594	The International Amateur Athletic Federation said on Friday that a schedule reshuffle had allowed organisers to hold a men s triple jump as well as the women s long jump on the one usable runway at the war-devastated Kosevo stadium .
595	Atlanta Games silver medal winner Edwards has called on other leading athletes to take part in the Sarajevo meeting -- a goodwill gesture towards Bosnia as it recovers from the war in the Balkans -- two days after the grand prix final in Milan .
596	Edwards was quoted as saying : What type of character do we show by going to the IAAF Grand Prix Final in Milan where there is a lot of money to make but refusing to make the trip to Sarajevo as a humanitarian gesture ?
598	SOCCER - BARATELLI TO COACH NICE .
599	NICE
600	Former international goalkeeper Dominique Baratelli is to coach struggling French first division side Nice
601	Baratelli
602	Nice have been unable to win any of their four league matches played this season and are lying a lowly 18th in the table .

**Rules: rule\_table**

type	regex
email	<code>[w\~]([\.w])+[w]+@[([w\~]+\.)+[a-zA-Z]{2,4}</code>

**SQL Call**

```

SELECT * FROM NERExtractor (
  ON ner_sports_test2 PARTITION BY ANY
  ON rule_table AS Rules DIMENSION
  USING
    TextColumn ('content')
    InputModelFiles ('ner_model.bin')
    ShowContext (2)
    Accumulate ('id')
) AS dt ORDER BY id, sn;

```

**Output**

id	sn	entity	type_ner	start_ner	end_ner	approach
context						
-----						
528	1	sports@espn.com	email	2	2	... email sports@espn.com
to contact		RULE				
529	1	cricket@espn.com	email	2	2	... email cricket@espn.com
to contact		RULE				
530	1	tennis@espn.com	email	2	2	... email tennis@espn.com
to contact		RULE				
531	1	Igor Trandenkov	PER	2	3	... 1= Igor Trandenkov
(Russia) 5.86		CRF				
532	1	Maksim Tarasov	PER	2	3	... 3. Maksim Tarasov
(Russia) 5.86		CRF				
533	1	Tim Lobinger	PER	2	3	... 4. Tim Lobinger
(Germany) 5.80		CRF				
534	1	Igor Potapovich	PER	2	3	... 5. Igor Potapovich
(Kazakstan) 5.80		CRF				
535	1	Jean Galfione	PER	2	3	... 6. Jean Galfione
(France) 5.65		CRF				
536	1	Pyotr Bochkary	PER	2	3	... 7. Pyotr Bochkary
(Russia) 5.65		CRF				
537	1	Dmitri Markov	PER	2	3	... 8. Dmitri Markov
(Belarus) 5.65		CRF				
583	1	GENEVA	LOC	1	1	... ... GENEVA
1996-08-30 ...		CRF				
584	1	Standard Liege	PER	8	9	Belgian club Standard
Liege on Friday		CRF				
587	1	Roberto Bisconti	PER	1	2	... ... Roberto Bisconti
will be		CRF				
591	1	HARRISON	PER	3	3	ATHLETICS -
HARRISON ... ...		CRF				
592	1	MONTE CARLO	PER	1	2	... ... MONTE CARLO
1996-08-30 ...		CRF				
593	1	Kenny Harrison	PER	3	4	Olympic champion Kenny
Harrison and world		CRF				
593	2	Jonathan Edwards	PER	9	10	record holder Jonathan
Edwards will both		CRF				
596	1	What	ORG	7	7	saying : What type
of		CRF				
598	1	BARATELLI TO	PER	3	4	SOCCER - BARATELLI TO
COACH NICE		CRF				

599	1	NICE	PER	1	1 ... ..
NICE	...	...		CRF	
600	1	Dominique Baratelli	PER	4	5 international goalkeeper
Dominique Baratelli	is	to	CRF		
600	2	Nice	PER	14	14 division side
Nice	...	...		CRF	
601	1	Baratelli	PER	1	1 ... ..
Baratelli	...	...		CRF	

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NEREvaluator

The NREvaluator function evaluates a CRF model (output by the function [NERTrainer](#)).

NEREvaluator uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NEREvaluator Syntax

### Version 1.9

```
SELECT * FROM NREvaluator (
  ON { table | view | (query) } PARTITION BY 1
  USING
  TextColumn ('text_column')
  ModelFile ('model_file[:jar_file]')
  [ InputLanguage ({ 'en' | 'zh_CN' | 'zh_TW' }) ]
) AS alias;
```

## NEREvaluator Syntax Elements

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### ModelFile

Specify the CRF model file to evaluate, created and automatically installed by [NERTrainer](#).

If you specified the ExtractorJAR syntax element in the NERTrainer call that created *model\_file*, then you must specify the same *jar\_file* in this syntax element. You must install the *jar\_file* on ML Engine before calling the NREvaluator function.

The names *model\_file* and *jar\_file* are case-sensitive.

**InputLanguage**

[Optional] Specify the language of the input text:

Option	Description
'en' (Default)	English
'zh_CN'	Simplified Chinese
'zh_TW'	Traditional Chinese

**NEREvaluator Input**

The input table has the same schema as the [NERExtractor Input](#) table.

**NEREvaluator Output****Output Table Schema**

Column	Data Type	Description
type	VARCHAR	Entity type. Final row value: -AVG-
precision	DOUBLE PRECISION	Precision value of the entity type. Final row value: Average precision value for all entity types.
recall	DOUBLE PRECISION	Recall value of the entity type. Final row value: Average recall value for all entity types.
f1_measure	DOUBLE PRECISION	F <sub>1</sub> score (F-measure) of the entity type. Final row value: Average F <sub>1</sub> score for all entity types.

**NEREvaluator Example**

This function evaluates the efficacy of the model file `ner_model.bin`, created by the `NERTrainer` function in terms of precision, recall, and `f1_measure`.

**Input**

- `ner_model.bin`, output by [NERTrainer Example](#)

**SQL Call**

```
SELECT * FROM NREvaluator (
  ON ner_sports_test2 PARTITION BY 1
```

```

USING
  TextColumn ('content')
  ModelFile ('ner_model.bin')
) AS dt;

```

## Output

```

type_ner precision_ner recall f1_measure
-----
LOC              1 0.4444      0.6154
ORG              0      0      -1
PER             0.7222 0.8125      0.7647
-AVG-           0.7778 0.4884      0.6

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NER Functions (Maximum Entropy Model Implementation)

Function	Description
<a href="#">NamedEntityFinderTrainer</a>	Takes training data and outputs a maximum entropy model (binary file).
<a href="#">NamedEntityFinder</a>	Evaluates input, identifies tokens based on specified model, and outputs tokens with detailed information. Uses model to extract entity types 'PERSON', 'LOCATION', and 'ORGANIZATION' and rules to extract entity types 'DATE', 'TIME', 'EMAIL' and 'MONEY'. If you specify these entity names, the function invokes the default model types and model file names. To extract all entities in one NamedEntityFinder call, specify 'ALL'.
<a href="#">Named Entity Finder Evaluator</a>	Evaluates maximum entropy model.

The maximum entropy model implementation supports only English text.

### Related Information:

[NER Functions \(CRF Model Implementation\)](#)

## NamedEntityFinderTrainer

The NamedEntityFinderTrainer function takes training data and outputs a Maximum Entropy data model. The function is based on OpenNLP, and follows its annotation. For more information on OpenNLP, see <https://opennlp.apache.org/docs/1.8.4/manual/opennlp.html>.

The trainer supports only the English language.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NamedEntityFinderTrainer Syntax

### Version 1.7

```
SELECT * FROM NamedEntityFinderTrainer (
  ON { table | view | (query) } PARTITION BY 1 [ ORDER BY order_column ]
  USING
  OutputModelFile (output_model_file)
  TextColumn ('text_column')
  EntityType ('entity_type')
  [ IterNum (iterator)]
  [ Cutoff (cutoff)]
) AS alias;
```

For repeatable results, you must specify ORDER BY and *order\_column* must have a unique value for each row.

## NamedEntityFinderTrainer Syntax Elements

### OutputModelFile

Specify the name of the data model file to create.

### TextColumn

Specify the name of the input table column that contains the text to analyze.

### EntityType

Specify the entity type to train (for example, PERSON). The input training documents must contain the same tag.

### IterNum

[Optional] Specify the iterator number for training (an openNLP training parameter).

Default: 100

### Cutoff

[Optional] Specify the cutoff number for training (an openNLP training parameter).

Default: 5

## NamedEntityFinderTrainer Input

### Input Table Schema

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax: <div>&lt;START:entity_type&gt;entity&lt;END&gt;</div> For example: <div>&lt;START:location&gt;Country1&lt;END&gt; has arrived</div>

## NamedEntityFinderTrainer Output

The function outputs a message and a Max Entropy model (a binary file automatically installed on ML Engine).

### Output Message Schema

Column	Data Type	Description
train_result	VARCHAR	Message indicating whether the function ran successfully.

## NamedEntityFinderTrainer Example

### Input

- Input Table: nermem\_sports\_train, which has 50 rows of sports news

#### Input Table: nermem\_sports\_train

id	content
2	CRICKET - <START:ORG> LEICESTERSHIRE <END> TAKE OVER AT TOP AFTER INNINGS VICTORY .
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
5	Their stay on top
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>

id	content
7	Trailing by 213
8	<START:ORG> Essex <END>
9	<START:PER> Hussain <END>
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderTrainer (
  ON nermem_sports_train PARTITION BY 1
  USING
    EntityType ('LOCATION')
    TextColumn ('content')
    OutputModelFile (location.sports)
) AS dt;
```

## Output

```
train_result
-----
model installed
```

The model table, location.sports, is in binary format.

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## NamedEntityFinder

The NamedEntityFinder function evaluates the input, identifies tokens based on the specified model, and outputs the tokens with detailed information. The function does not identify sentences; it simply tokenizes. Token identification is not case-sensitive.

NamedEntityFinder uses files that are preinstalled on ML Engine. For details, see [Preinstalled Files That Functions Use](#).

## NamedEntityFinder Syntax

**Version 1.6**

```

SELECT * FROM NamedEntityFinder (
  ON { table | view | (query) } PARTITION BY ANY
  [ ON (configure_table) AS ConfigurationTable DIMENSION ]
  USING
  TextColumn ('text_column')
  [ Models ('entity_type[:model_type:{model_file|regular_expression}')[, ...] |
  'all' }) ]
  [ ShowContext ('context_words') ]
  [ EntityColName ('entity_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[, ...]) ]
) AS alias;

```

**Related Information:**

[Column Specification Syntax Elements](#)

[Regular Expressions in Syntax Elements](#)

**NamedEntityFinder Syntax Elements****TextColumn**

Specify the name of the input table column that contains the text to analyze.

**Models**

[Optional] Required if you do not specify ConfigurationTable, in which case you cannot specify 'all'. Specify the model items to load.

If you specify both ConfigurationTable and this syntax element, the function loads the specified model items from ConfigurationTable.

The *entity\_type* is the name of an entity type (for example, PERSON, LOCATION, or EMAIL), which appears in the output table.

<i>model_type</i>	Description
'max entropy'	Maximum entropy language model output by training.
'rule'	Rule-based model, a plain text file with one regular expression on each line.
'dictionary'	Dictionary-based model, a plain text file with one word on each line.
'reg exp'	Regular expression that describes <i>entity_type</i> .

If *model\_type* is 'reg exp', specify *regular\_expression* (a regular expression that describes *entity\_type*); otherwise, specify *model\_file* (the name of the model file).

If you specify ConfigurationTable, you can use *entity\_type* as a shortcut. For example, if the ConfigurationTable has the row 'organization, max entropy, en-ner-organization.bin', you can

specify Models ('organization') as a shortcut for Models ('organization:max entropy:en-ner-organization.bin').

---

**Note:**

For *model\_type* 'max entropy', if you specify ConfigurationTable and omit this syntax element, then the JVM of the worker node needs more than 2GB of memory.

---

Default: 'all' (If you specify ConfigurationTable but omit this syntax element.)

**ShowContext**

[Optional] Specify the number of context words to output. If *context\_words* is *n* (which must be a positive integer), the function outputs the *n* words that precede the entity, the entity, and the *n* words that follow the entity.

Default: 0

**EntityColName**

[Optional] Specify the name of the output table column that contains the entity names.

Default: 'entity'

**Accumulate**

[Optional] Specify the names of input columns to copy to the output table. No *accumulate\_column* can be an *entity\_column*.

Default: All input columns

## Creating the Table of Default Models

Before calling the NamedEntityFinder function, you must create the table of default models. To create the table, use this command:

```
DROP TABLE nameFind_configure;

CREATE MULTISET TABLE nameFind_configure (
  model_name VARCHAR(50),
  model_type VARCHAR(50),
  model_file VARCHAR(50)
);
```

Default English-language models are provided with the SQL functions. Before using these models, you must create a default *configure\_table*, as follows:

```
INSERT INTO nameFind_configure VALUES ('person','max entropy','en-ner-person.bin');
INSERT INTO nameFind_configure VALUES ('location','max entropy','en-ner-location.bin');
INSERT INTO nameFind_configure VALUES ('organization','max entropy','en-ner-
```

```
organization.bin');
INSERT INTO nameFind_configure VALUES ('date','rules','date.rules');
INSERT INTO nameFind_configure VALUES ('time','rules','time.rules');
INSERT INTO nameFind_configure VALUES ('phone','rules','phone.rules');
INSERT INTO nameFind_configure VALUES ('money','rules','money.rules');
INSERT INTO nameFind_configure VALUES ('email','rules','email.rules');
INSERT INTO nameFind_configure VALUES ('percentage','rules','percentage.rules');
```

#### Default English-Language Models in Table nameFind\_configure

model_name	model_type	model_file
person	max entropy	en-ner-person.bin
location	max entropy	en-ner-location.bin
organization	max entropy	en-ner-organization.bin
date	rules	date.rules
time	rules	time.rules
phone	rules	phone.rules
money	rules	money.rules
email	rules	email.rules
percentage	rules	percentage.rules

## NamedEntityFinder Input

### Input Table Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Contains input text.
<i>accumulate_column</i>	Any	Column to copy to output table.

### ConfigurationTable Schema

This table is optional.

Column	Data Type	Description
<i>model_name</i>	VARCHAR	Name of an entity type (for example, PERSON, LOCATION, or EMAIL).
<i>model_type</i>	VARCHAR	One of these model types:

Column	Data Type	Description										
		<table><tr><th><i>model_type</i></th><th>Description</th></tr><tr><td>'max entropy'</td><td>Maximum entropy language model created by training</td></tr><tr><td>'rule'</td><td>Rule-based model, a plain text file with one regular expression on each line</td></tr><tr><td>'dictionary'</td><td>Dictionary-based model, a plain text file with one word on each line</td></tr><tr><td>'reg exp'</td><td>Regular expression that describes <i>entity_type</i></td></tr></table>	<i>model_type</i>	Description	'max entropy'	Maximum entropy language model created by training	'rule'	Rule-based model, a plain text file with one regular expression on each line	'dictionary'	Dictionary-based model, a plain text file with one word on each line	'reg exp'	Regular expression that describes <i>entity_type</i>
<i>model_type</i>	Description											
'max entropy'	Maximum entropy language model created by training											
'rule'	Rule-based model, a plain text file with one regular expression on each line											
'dictionary'	Dictionary-based model, a plain text file with one word on each line											
'reg exp'	Regular expression that describes <i>entity_type</i>											
<i>model_file</i>	VARCHAR	Name of model file that describes the entity type. This column appears if <i>model_type</i> is not 'reg exp'.										
reg_exp	VARCHAR	Regular expression that describes the entity type. This column appears if <i>model_type</i> is 'reg exp'.										

## NamedEntityFinder Output

### Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
entity_type	VARCHAR	Entity type.
entity	VARCHAR	Entity name.
entity_start	INTEGER	[Column appears only with ShowEntityContext syntax element.] Start position.
entity_end	INTEGER	[Column appears only with ShowEntityContext syntax element.] End position.
context	VARCHAR	[Column appears only with ShowEntityContext syntax element.] Words before and after the entity.

## NamedEntityFinder Example

### Input

Input Table: assortedtext\_input

id	source	content
1001	misc	contact Alan by email at sports@espn.com for all sport info

id	source	content
1002	misc	contact Mark at cricket@espn.com for all cricket info
1003	misc	contact Roger at tennis@espn.com for all tennis info
1004	wiki	The contiguous United States consists of the 48 adjoining U.S. states plus Washington, D.C., on the continent of North America
1005	wiki	California's economy is centered onTechnology,Finance,real estate services, Government, and professional, Scientific and Technical business Services; together comprising 58% of the State Government economy
1006	wiki	Houston is the largest city in Texas and the fourth-largest in the United States, while San Antonio is the second largest and seventh largest in the state.
1007	wiki	Thomas is a photographer whose natural landscapes of the West are also a statement about the importance of the preservation of the wildness

## SQL Call

```
SELECT * FROM NamedEntityFinder (
  ON assortedtext_input PARTITION BY ANY
  ON namefind_configure AS ConfigurationTable DIMENSION
  USING
    TextColumn ('content')
    Models ('all')
    Accumulate ('id', 'source')
) AS dt ORDER BY id;
```

## Output

id	source	entity	entity_type
----	-----	-----	-----
1001	misc	sports@espn.com	email
1002	misc	cricket@espn.com	email
1002	misc	Mark	person
1003	misc	Roger	person
1003	misc	tennis@espn.com	email
1004	wiki	Washington	location
1004	wiki	U.S.	location
1004	wiki	North America	location
1004	wiki	United States	location
1005	wiki	State Government	organization
1005	wiki	58%	percentage
1006	wiki	San Antonio	location
1006	wiki	United States	location

1006 wiki	Texas	location
1007 wiki	Thomas	person

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Named Entity Finder Evaluator

The `NamedEntityFinderEvaluatorMap` and `NamedEntityFinderEvaluatorReduce` functions operate as a row and a partition function, respectively. Each function takes a set of evaluating data and creates the precision, recall, and F-measure values of a specified maximum entropy data model. Neither function supports regular-expression-based or dictionary-based models.

### Related Information:

[Nondeterministic Results and UniqueID Syntax Element](#)

## Named Entity Finder Evaluator Syntax

### **NamedEntityFinderEvaluatorReduce version 1.5, NamedEntityFinderEvaluatorMap version 1.7**

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON { table | view | (query) }
    USING
    TextColumn ('text_column')
    InputModelFile ('input_model_file')
  ) AS alias_1 PARTITION BY 1
) AS alias_2;
```

## Named Entity Finder Evaluator Syntax Elements

### **TextColumn**

Specify the name of the input table column that contains the text to analyze.

### **InputModelFile**

Specify name of the model file to evaluate.

## NamedEntityFinderEvaluatorMap Input

### Input Table Schema

Column	Data Type	Description
<i>text_column</i>	VARCHAR	Text to analyze. Within the text, each entity must be identified with this syntax: <div>&lt;START:entity_type&gt; entity &lt;END&gt;</div> For example: <div>&lt;START:location&gt;Country1&lt;END&gt; has arrived</div>

## NamedEntityFinderEvaluatorReduce Output

### Output Table Schema

Column	Data Type	Description
precision_val	INTEGER	Precision value of the model.
recall	DOUBLE PRECISION	Recall value of the model.
f_measure	DOUBLE PRECISION	F-measure ( $F_1$ score) of the model.

## Named Entity Finder Evaluator Example

### Input

- Input Table: nermem\_sports\_test, which has rows of sports news
- model\_file: location.sports, output by [NamedEntityFinderTrainer Example](#)

#### Input Table: nermem\_sports\_test

id	content
3	<START:LOCATION> LONDON <END> 1996-08-30
4	West Indian all-rounder <START:PER> Phil Simmons <END> took four for 38 on Friday as <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs in two days to take over at the head of the county championship .
6	After bowling <START:ORG> Somerset <END> out for 83 on the opening morning at <START:LOCATION> Grace Road <END>
9	<START:PER> Hussain <END>

id	content
10	By the close <START:ORG> Yorkshire <END> had turned that into a 37-run advantage but off-spinner <START:PER> Such <END> had scuttled their hopes
11	At the <START:LOCATION> Oval <END>
12	He was well backed by <START:LOCATION> England <END> hopeful <START:PER> Mark Butcher <END> who made 70 as <START:ORG> Surrey <END> closed on 429 for seven
14	Australian <START:PER> Tom Moody <END> took six for 82 but <START:PER> Chris Adams <END>
16	They were held up by a gritty 84 from <START:PER> Paul Johnson <END> but ex-England fast bowler <START:PER> Martin McCague <END> took four for 55 .
20	<START:LOCATION> LONDON <END> 1996-08-30
22	<START:LOCATION> Leicester <END> : <START:ORG> Leicestershire <END> beat <START:ORG> Somerset <END> by an innings and 39 runs .
...	...

## SQL Call

```
SELECT * FROM NamedEntityFinderEvaluatorReduce (
  ON NamedEntityFinderEvaluatorMap (
    ON nermem_sports_test
    USING
    InputModelFile ('location.sports')
    TextColumn ('content')
  ) PARTITION BY 1
) AS dt;
```

## Output

```
precision_val    recall          f_measure
-----
0.847457627118644 0.7936507936507936 0.819672131147541
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.



## Module 03 –

# Pattern Detection with nPath

Teradata Vantage Analytics Workshop BASICS

©2019 Teradata

# Objectives

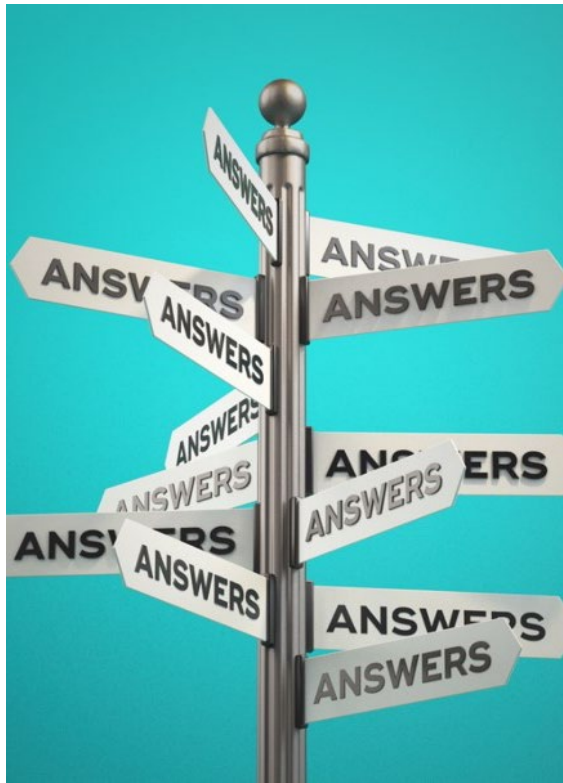
After completing this module, you will be able to:

- Describe what the **nPath** function does
- Describe typical use cases for **nPath**
- Write **nPath** queries
- Interpret the output of **nPath** queries
- Run an **nPath** visualization in Teradata AppCenter

For more info go to [docs.teradata.com](https://docs.teradata.com) click Teradata Vantage, download: Teradata Vantage Machine Learning Engine Analytic Function Reference guide.

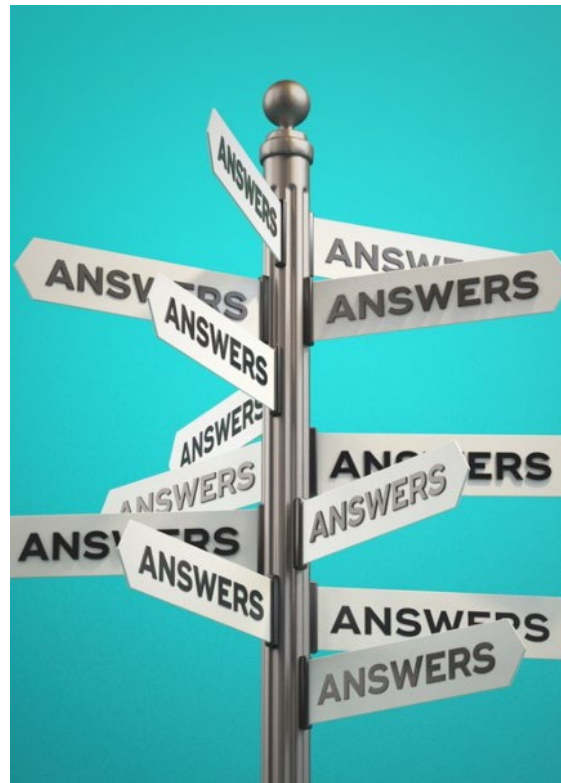
# Topics

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- Symbols
- Mode
- Pattern and Symbols
- Results
- Teradata AppCenter
- Hackathon and Review



# Current Topic – Background Information

- **Background Information**
  - **Description**
  - **Use Cases**
  - **Syntax**
  - **Input Data**
  - **Required Arguments**
  - **Optional Arguments**
  - **Input Table Schema**
- Symbols
- Mode
- Pattern and Symbols
- Results
- Teradata AppCenter
- Hackathon and Review





# Description

## What is nPath?

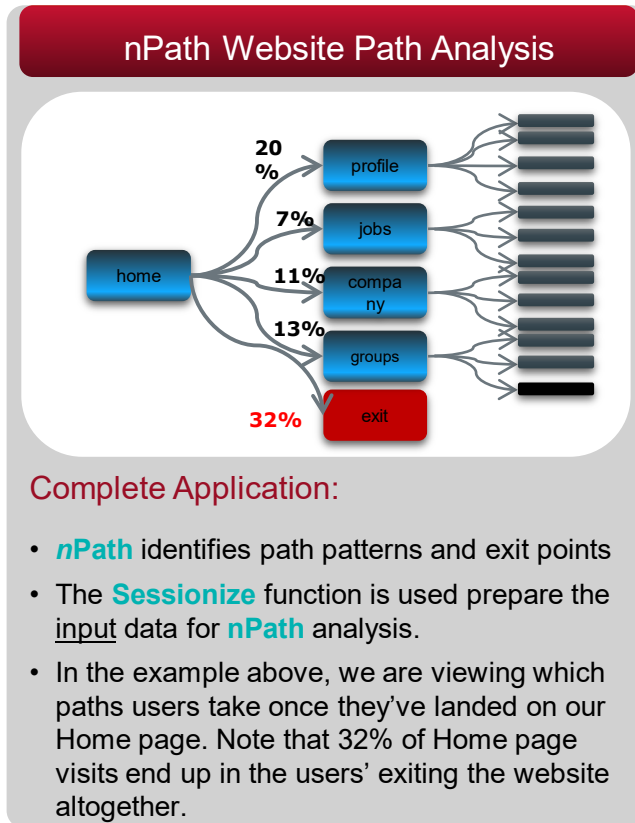
- Function designed for time-series sequence analysis of data
- Links an outcome with a preceding path

## Benefits

- *Pattern detection can be completed in a single pass over the data*
- Allows you to understand relationships across rows of data
- Transcends SQL's ordered-data limitations that require either complex, multi-pass SQL or custom UDFs for each analysis

## Example use cases:

- Web analytics (clickstream, Golden Path)
- Complex Marketing revenue paths
- Granular product & process analysis (A/B)
- Granular pattern detection (fraud, QA,..)





## nPath Use Cases

7

Some examples of how **nPath** can be used follow:

- A **Retailer** wishes to analyze Web site click data, to identify paths that lead to sales over a specified amount
- A **Manufacturer** analyzes sensor data FROM industrial processes, to identify paths to poor product quality
- A **Healthcare provider** analyzes healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- A **Financial institution** reviews financial data for individuals, to identify paths that provide information about credit or fraud risks



# nPath Workflow



- **Input Tables(s):** Data is read FROM specified input tables, views, or queries
- **nPath:** The following arguments are specified when the function is invoked
  - **Mode (overlapping or nonoverlapping)**
  - **Pattern to match**
  - **Symbols to use**
  - **[Optional] Filters to apply**
  - **Results to output**
- **Output table:** Data is written to an output table



# nPath Syntax

```
SELECT * FROM nPath [@coprocess]
(ON { table | view | (query) }
PARTITION BY partition_column
ORDER BY order_column[ ASC | DESC ]
[ ON { table | view | (query) }
[ PARTITION BY partition_column | DIMENSION ] ORDER BY column [ ASC | DESC ]]
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])
) AS dt;
```



# Lab 01 – nPath Simple Query

## Input Data

```
SELECT * FROM borre_z  
ORDER BY ts;
```

	user id	event	ts
1	1	a	2017-01-01 13:21:01.000000
2	1	a	2017-01-01 13:21:02.000000
3	1	a	2017-01-01 13:21:03.000000
4	1	a	2017-01-01 13:21:04.000000

- On the following pages, we will discuss the required arguments for **nPath**
- For each required argument, we will discuss the implications of our specifications using the simple **nPath** query to the right as the foundation

## nPath Query

```
SELECT * FROM nPath@coprocessor  
(ON borre_z  
PARTITION BY user_id  
ORDER BY ts  
USING  
Mode (NONOVERLAPPING)  
Pattern ('X.X')  
Symbols (event = 'a' as X)  
Result (Accumulate (event OF X) AS x_pattern)  
) AS dt;
```

Remove '@coprocessor  
and run again using MLE

**Note:** In the query above, the **@coprocessor** tag signifies that we are opting to run the query on the Machine Learning Engine. Without this tag, the query would run on the Advanced SQL Engine. For this module's labs, we will run our queries on the Advanced SQL Engine (i.e., without the **@coprocessor** tag)

## nPath Results

	x pattern
1	[a, a]
2	[a, a]



## nPath Input Data (1 of 3)

```
SELECT * FROM nPath@coprocessor (  
ON { table | view | (query) } PARTITION BY partition_column ORDER BY order_column  
[ ASC | DESC ]  
[ ON { table | view | (query) }  
[ PARTITION BY partition_column | DIMENSION ] ORDER BY order_column [ ASC | DESC ]  
)[...]
```

Here, we specify our input data.

- The **FROM** keyword is followed by **nPath** (or **nPath@coprocessor**). This invokes the **nPath** function
- The **ON** keyword is followed by our input data (**borre\_z**)
- We **PARTITION BY** **user\_id** in this example and **ORDER BY** **ts**

### nPath Query

```
SELECT * FROM nPath  
(ON borre_z  
PARTITION BY user_id  
ORDER BY ts  
USING  
Mode (NONOVERLAPPING)  
Pattern ('X.X')  
Symbols (event = 'a' as X)  
Result (Accumulate (event OF X) AS x_pattern)  
) AS dt;
```



## nPath Input Data (2 of 3)

12

```
... ON (SELECT * FROM WEBCLICKS)  
PARTITION BY user_id, sessionid  
ORDER BY timestamp ...
```

Partition for User\_Id 1 Session\_Id 0    Partition for User\_Id 1 Session\_Id 1    Partition for User\_Id 97 Session\_Id 0

User_Id	Session_Id	Date stamp	Other Columns
1	0	2011-01-02 02:15:00	Home page, etc,
1	0	2011-01-02 02:16:00	Product page, etc
1	0	2011-01-02 02:17:00	Profile page, etc
1	0	2011-01-02 02:18:00	Watch page, etc
1	0	2011-01-02 02:19:00	Logout page, etc

User_Id	Session_Id	Date stamp	Other Columns
1	1	2011-01-11 09:27:00	Home page, etc,
1	1	2011-01-11 09:28:00	Product page, etc
1	1	2011-01-11 09:30:00	Product page, etc
1	1	2011-01-11 09:32:00	Bid, etc

User_Id	Session_Id	Date stamp	Other Columns
97	0	2011-03-13 07:17:00	Home page, etc,
97	0	2011-03-13 07:18:00	Coupon page, etc
97	0	2011-03-13 07:19:00	Customer support page, etc
97	0	2011-03-13 07:21:00	Sell page, etc
97	0	2011-03-13 07:23:00	History page, etc

**PARTITION BY** groups rows with like values together.

**ORDER BY** then sorts each partition according to our specifications.

(Now, the clicks of each User\_Id are sorted in the sequence in which they were made)



## nPath Input Data (3 of 3)

- **ON expression**

- The input Table, View, or Query

- **PARTITION BY expression** [...]

- The attribute(s) by which the rows are grouped
- Identifies entity of interest – such as user\_id, product\_id, etc.

- **ORDER BY expression** [ASC|DESC] [...]

- The expression by which the rows within each partition are ordered
- Typically a date/time field, but can be any sequence attribute

Selecting all rows from savings

```
SELECT * FROM nPath
(ON savings
PARTITION BY cust
ORDER BY ts
..... )
```

Selecting a subset of rows from savings

```
SELECT * FROM nPath
(ON (SELECT * FROM savings
WHERE amt > 0) as t1
PARTITION BY cust
ORDER BY ts
..... )
```



## nPath Required Arguments: Mode

USING

Mode ({ **OVERLAPPING** | **NONOVERLAPPING** })

Pattern ('pattern')

Symbols ({ col\_expr = symbol\_predicate **AS** symbol } [,...])

[ Filter (filter\_expression [,...]) ]

Result ({ aggregate\_function (col\_expr **OF** symbol) **AS** alias\_1 } [,...])  
) **AS** alias\_2;

Here, we specify the pattern-matching mode.

There are two flavors of this:

- **OVERLAPPING**: Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern
- **NONOVERLAPPING**: Start next pattern search at row that follows last pattern match

**SELECT \* FROM** **nPath**

(**ON** borre\_z

**PARTITION BY** user\_id **ORDER BY** ts

**USING**

**Mode (NONOVERLAPPING)**

Pattern ('X.X')

Symbols (event = 'a' **as** X)

Result (Accumulate (event **OF** X) **AS** x\_pattern)  
) **AS** dt;



## nPath Required Arguments: Pattern

```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 } [,...])
) AS alias_2;
```

Here, we specify the **Pattern** for which the function searches.

- We compose **Pattern** with symbols (which we define in the **Symbols** argument), operators, and parentheses. Here, we are searching for *Symbol X followed by X* (which means event 'a' followed by event 'a')
- When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence FROM left to right. To force the function to evaluate a subpattern first, enclose it in parentheses

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Mode (NONOVERLAPPING)
Pattern ('X.X')
Symbols (event = 'a' as X)
Result (Accumulate(event OF X) AS x_pattern)
) AS dt;
```



## nPath Required Arguments: Symbols

```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 } [,...])
) AS alias_2;
```

Here, we specify the **Symbols** that appear in the values of the **Pattern** and **Result** arguments

- The *col\_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol\_predicate* is a SQL predicate (often a column name)
- You can think of **Symbols** as the 'aliases' that you will be defining for use in the **Pattern** and **Result** portions of the query

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Mode (NONOVERLAPPING)
Pattern ('X.X')
Symbols (event = 'a' as X)
Result (Accumulate (event OF X) AS x_pattern)
) AS dt;
```



## nPath Required Arguments: Result

```
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
[ Filter (filter_expression [,...]) ]
Result ({ aggregate_function (col_expr OF symbol) AS alias_1 } [,...])
) AS alias_2;
```

Here, we specify the output columns

- The *col\_expr* is an expression whose value is a column name; it specifies the values to retrieve FROM the matched rows. The function applies *aggregate\_function* to these values
- The function evaluates this argument once for every matched pattern in the partition (that is, it outputs one row for each pattern match)

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Mode (NONOVERLAPPING)
Pattern ('X.X')
Symbols (event = 'a' as X)
Result (Accumulate (event OF X) AS x_pattern)
) AS dt;
```



## Lab 02 – Changing the Order of Required Arguments doesn't make a Difference

- You can change the order of arguments that appear after **USING** as desired
- For example, you may find it more logical to define **Symbols** right away
- All of the following queries will return the same answer-set

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Symbols (event = 'a' as X)
Pattern ('X.X')
Mode (NONOVERLAPPING)
Result (Accumulate (event OF
X) AS x_pattern)) AS dt;
```

	x pattern
1	[a, a]
2	[a, a]

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Pattern ('X.X')
Mode (NONOVERLAPPING)
Symbols (event = 'a' as X)
Result (Accumulate (event OF
X) AS x_pattern)) AS dt;
```

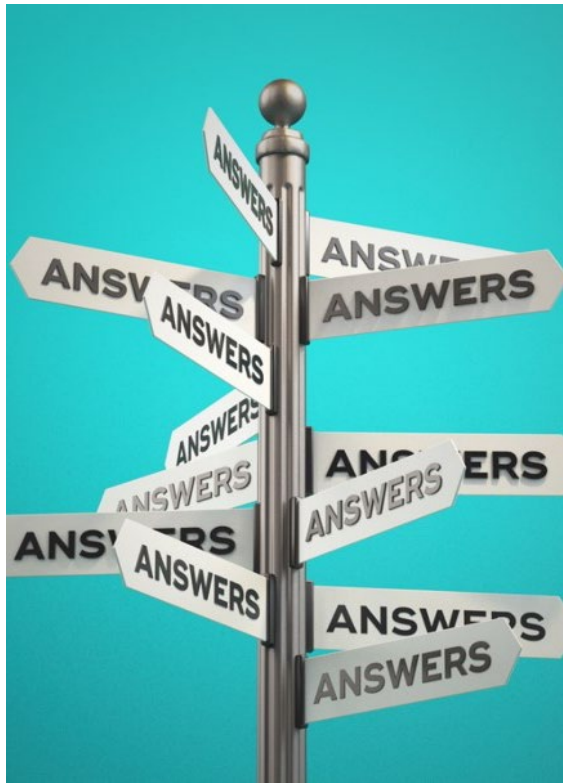
	x pattern
1	[a, a]
2	[a, a]

```
SELECT * FROM nPath
(ON borre_z
PARTITION BY user_id
ORDER BY ts
USING
Pattern ('X.X')
Symbols (event = 'a' as X)
Mode (NONOVERLAPPING)
Result (Accumulate (event OF
X) AS x_pattern)) AS dt;
```

	x pattern
1	[a, a]
2	[a, a]

# Current Topic – Symbols

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- **Symbols**
  - Mode
  - Pattern and Symbols
  - Results
  - Teradata AppCenter
  - Hackathon and Review





## Lab 03 – Symbols Example

- Recall that the **Symbols** argument allows us to define the aliases that we wish to use in the **Pattern** and **Results** arguments
- The next few pages will walk through some very straightforward examples that should illustrate how **Symbols** work
- For all examples, assume four-row table appearing below
- For all examples, we will be searching for a product of **apple** followed by **banana**, using whatever **Symbols** we may have decided to define
- In our dataset, the only pattern match will be rows 1 (apple) and 2 (banana)

```
SELECT *  
FROM borre_food  
ORDER BY event_id;
```

	user name	event id	product
1	tom	0	apple
2	tom	1	banana
3	tom	2	cherry
4	tom	3	date



## Lab 03 – Symbols Example

```
SELECT products_accumulate, count (*)
FROM nPath
(ON borre_food
PARTITION BY user_name
ORDER BY event_id
USING
Mode (NONOVERLAPPING)
Pattern ('a.b')
Symbols(product = 'apple' as a,
         product = 'banana' as b)
Result (
ACCUMULATE (product OF ANY (a,b)
DELIMITER '*')
AS products_accumulate)
) AS dt
GROUP BY products_accumulate;
```

- Here, we are defining **Symbols** of **a** for apple and **b** for banana
- Our defined **Symbols** are used in the **Pattern** and **Result** arguments

### Input

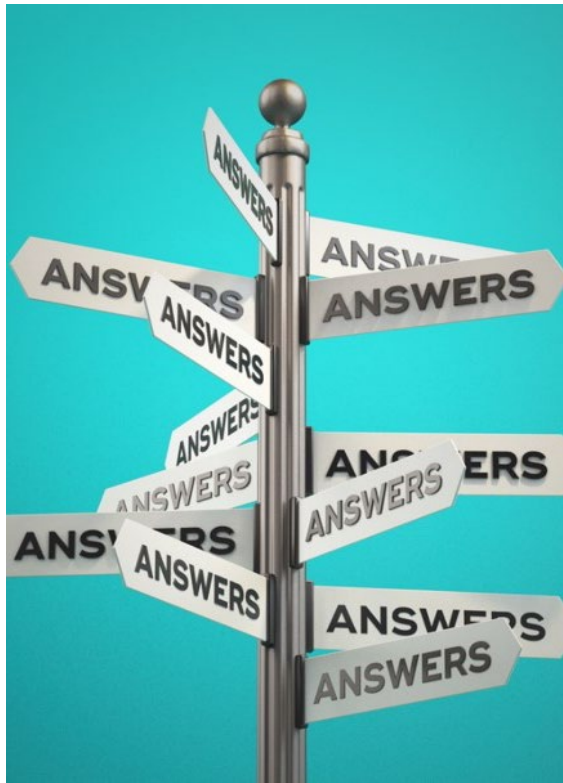
	user name	event id	product
1	tom	0	apple
2	tom	1	banana
3	tom	2	cherry
4	tom	3	date

### Output

	products accumulate	Count(*)
1	[apple*banana]	1

# Current Topic – Mode

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- Symbols
- **Mode**
- Pattern and Symbols
- Results
- Teradata AppCenter
- Hackathon and Review





## Lab 04 – Mode Example (1 of 3)

- Recall that the **Mode** argument can have a value of **NONOVERLAPPING** or **OVERLAPPING**
- The next few pages will walk through how the **Mode** value that you specify will impact the answer-set
- For the example, assume a simple input table with two columns and four rows

```
SHOW TABLE matchup;
```

```
SELECT *  
FROM matchup  
ORDER BY c2, c1;
```

```
CREATE MULTISET TABLE matchup ,FALLBACK ,  
  NO BEFORE JOURNAL,  
  NO AFTER JOURNAL,  
  CHECKSUM = DEFAULT,  
  DEFAULT MERGEBLOCKRATIO,  
  MAP = TD_MAP1  
  (  
    c1 INTEGER,  
    c2 VARCHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC)  
  PRIMARY INDEX ( c1 );
```

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A



## Lab 04 – Mode Example (2 of 3)

**PATTERN(A.A) = Search for A followed by A**

In **NONOVERLAPPING** match mode, **nPath** begins the next pattern search at the row that follows the last row that was part of the previous **PATTERN** match. In this example, the next pattern match starts at row 3

```
SELECT * FROM nPath
(ON matchup
PARTITION BY c2
ORDER BY c1
USING
Mode (NONOVERLAPPING)
Pattern ('A.A')
Symbols (c2 = 'A' as A)
Result
(Accumulate (c2 OF A) AS x_pattern))
AS dt;
```

Input

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A

Output

	x pattern
1	[A, A]
2	[A, A]

After you have a complete Pattern match (TRUE), assuming there are more rows in the input table:  
If **NONOVERLAPPING**, start next Pattern match on next row after last matched row



## Lab 04 – Mode Example (3 of 3)

**PATTERN(A.A) = Search for A followed by A**

In **OVERLAPPING** match mode, **nPath** finds every occurrence of the pattern, regardless of whether it might have been part of a previously found match. This means that, in **OVERLAPPING** mode, one row can match multiple symbols in a given matched **PATTERN**

```
SELECT * FROM nPath
(ON matchup
PARTITION BY c2
ORDER BY c1
USING
Mode (OVERLAPPING)
Pattern ('A.A')
Symbols (c2 = 'A' as A)
Result
(Accumulate (c2 OF A) AS x_pattern)
) AS dt;
```

Input

	c1	c2
1	1	A
2	2	A
3	3	A
4	4	A

Output

	x pattern
1	[A, A]
2	[A, A]
3	[A, A]

After you have a complete **Pattern** match (TRUE), assuming there are more rows in the input table:

If **OVERLAPPING**, go back to the second matched row of the previous matched Pattern. Repeat as needed until no more matches. If/when have no more matches, start next Pattern match on row after last matched row

# Current Topic – Pattern and Symbols

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- Symbols
- Mode
- **Pattern and Symbols**
- Results
- Teradata AppCenter
- Hackathon and Review





# Pattern Operators (1 of 2)

- Use with pattern symbols to customize pattern-matching rules
  - '.' : followed by (Use to separate a series of pattern symbols)
  - '|' : alternative (The equivalent of an OR)
  - '?' : occurs at most once (0-1)
  - '\*' : occurs zero or more times (0-n)
  - '+' : occurs at least once (1-n)
  - '^' : pattern must begin with value specified. Also, value specified must be the first row within the partition.
  - '\$' : pattern must end with
- Customizing pattern matching rules:
  - (X){a}: exactly A number of occurrences of X `pattern('A.B{3}')`
  - (X){a,}: at least A number of occurrences of X `pattern('A.B{1,}')`
  - (X){a,b}: A to B occurrences of X `pattern('A.B{1,3}')`

## Pattern Operators (2 of 2)

Operator	Description	Precedence
<b>A</b>	Matches one row that meets the definition of A	1 (highest)
<b>A.</b>	Matches one row that meets the definition of A	1
<b>A?</b>	Matches 0 or 1 rows that satisfy the definition of A	1
<b>A*</b>	Matches 0 or more rows that satisfy the definition of A (greedy operator)	1
<b>A+</b>	Matches 1 or more rows that satisfy the definition of A (greedy operator)	1
<b>A.B</b>	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B	2
<b>A B</b>	Matches one row that meets the definition of either A or B	3

The **nPath** function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators

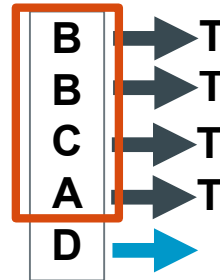


## Lab 06 – Walking the Rows on Pattern Given Mode

```
SELECT * FROM nPath
(ON npathBetween2
PARTITION BY c2
ORDER BY c1
USING Mode (NONOVERLAPPING)
Pattern ('B+.C.A')
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

```
SELECT * FROM nPath
(ON npathBetween2
PARTITION BY c2
ORDER BY c1
USING Mode (OVERLAPPING)
Pattern ('B+.C.A')
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Match 1 of 1



With **nonoverlapping**, after the first match, we start the next pattern on the **fifth** row

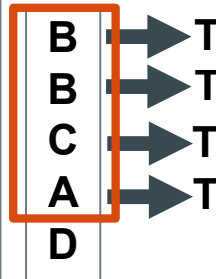
Input

c1	c2	c3
1	1	B
2	1	B
3	1	C
4	1	A
5	1	D

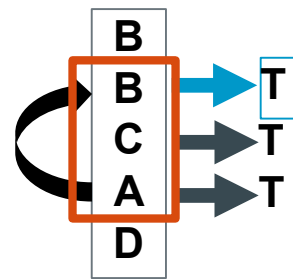
matches

[B, B, C, A]

Match 1 of 2



Match 2 of 2



matches

[B, C, A]  
[B, B, C, A]

With **overlapping**, after the first match, we start the next pattern on the **second** row



## Lab 07 – Followed By ( . )

```
SELECT * FROM nPath
(ON npathBetween2
PARTITION BY c2
ORDER BY c1
USING Mode (NONOVERLAPPING)
Pattern ('B.C.A'))
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Here, we are searching for **B followed by C followed by A**

Input

c1	c2	c3
1	1	B
2	1	B
3	1	C
4	1	A
5	1	D

B  
B  
C  
A  
D

Output

	matches
1	[B, C, A]



## Lab 08 – OR ( | )

```
SELECT * FROM nPath
(ON npathBetween2
PARTITION BY c2
ORDER BY c1
USING Mode (NONOVERLAPPING)
Pattern ('B|C|A'))
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Here, we are searching for any of the following

- B, OR
- C, OR
- A

B
B
C
A
D

Input

c1	c2	c3
1	1	B
2	1	B
3	1	C
4	1	A
5	1	D

Output

	matches
1	[B]
2	[B]
3	[C]
4	[A]



## Lab 09 – Followed By Together With OR

```
SELECT * FROM nPath
(ON npathBetween2
PARTITION BY c2
ORDER BY c1
USING Mode (NONOVERLAPPING)
Pattern ('B.C|A'))
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

Here, we are searching for either of the following:

- *B followed by C, OR*
- *A*

Input

c1	c2	c3
1	1	B
2	1	B
3	1	C
4	1	A
5	1	D

B
B
C
A
D

Output

	matches
1	[B, C]
2	[A]



## Lab 10 – Parentheses (1 of 2)

```
SELECT * FROM nPath
```

```
(ON npathBetween2
```

```
PARTITION BY c2
```

```
ORDER BY c1
```

```
USING Mode (NONOVERLAPPING)
```

```
Pattern ('B.(C|A)')
```

```
Symbols (c3='A' AS A, c3='B' AS B, c3='C' AS C)
```

```
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)  
) AS dt;
```

- Here, we are searching for a **B**, followed by either a **C** or an **A**
- Note that absent parentheses, FOLLOWED BY [ . ] takes precedence over OR [ | ]
- Be aware of orders of operation and how the presence or absence of parentheses may impact your answer-sets

Input

c1	c2	c3
1	1	B
2	1	B
3	1	C
4	1	A
5	1	D

B  
B  
C  
A  
D

Output

	matches
1	[B, C]



## Lab 10 – Parentheses (2 of 2)

### No Parentheses

```
SELECT * FROM nPath
(ON npathBetween
PARTITION BY c2
ORDER BY c1
USING
Mode (NONOVERLAPPING)
Pattern ('A.B|C')
Symbols (c3='A' AS A,c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

### Input

c1	c2	c3
1	1	A
2	1	B
3	1	C
4	1	A
5	1	B
6	1	A
7	1	C

### Output

	matches
1	[A, B]
2	[C]
3	[A, B]
4	[C]

### Parentheses

```
SELECT * FROM nPath
(ON npathBetween
PARTITION BY c2
ORDER BY c1
USING
Mode (NONOVERLAPPING)
Pattern ('A.(B|C)')
Symbols (c3='A' AS A,c3='B' AS B, c3='C' AS C)
Result (Accumulate(c3 of ANY(B,C,A)) AS Matches)
) AS dt;
```

### Input

c1	c2	c3
1	1	A
2	1	B
3	1	C
4	1	A
5	1	B
6	1	A
7	1	C

### Output

	matches
1	[A, B]
2	[A, B]
3	[A, C]



## Lab 11 – Exploring the ^ Predicate (1 of 3)

- Recall that the ^ predicate forces the **Pattern** to begin with whatever you specify. Furthermore, it forces the value specified to be in the first row of the partition
- This lab will focus on the input data displayed below
- We will show an example of each of the following for our **Pattern**:
  - Pattern ('sw.next\_job')**
  - Pattern ('^sw.next\_job')**

```
SELECT *  
FROM jobs  
ORDER BY emp_id, bgn_dt;
```

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31



## Lab 11 – Without ^ Predicate (2 of 3)

```
SELECT emp_id, Matches, count(*)
FROM nPath@coprocessor
(ON jobs
PARTITION BY emp_id
ORDER BY bgn_dt
USING
Mode (NONOVERLAPPING)
Pattern ('sw.next_job')
Symbols (job_desc ilike '%soft%'
as sw, TRUE as next_job)
Result (
First (emp_id of sw) as emp_id,
Accumulate(job_desc of
any(sw,next_job)) as Matches)
) AS dt
GROUP BY emp_id, Matches
ORDER BY emp_id;
```

- Here, we are not specifying the ^ predicate in our **Pattern** argument
- Given this, we're searching for any job-path pattern that goes FROM '**%soft%**' to anything else, whatever it is
- Both **emp\_id** values have met our conditions

### Input

	emp id	job desc	bgn dt	end dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31

### Output

	emp id	matches	Count(*)
1	1	[Software Engineer, Director of Engineering]	1
2	2	[Software Engineer, Director of Engineering]	1



## Lab 11 – With ^ Predicate (3 of 3)

```
SELECT emp_id, Matches, count(*)
FROM nPath@coprocessor
  (ON jobs
  PARTITION BY emp_id
  ORDER BY bgn_dt
  USING
  Mode (NONOVERLAPPING)
  Pattern ('^sw.next_job')
  Symbols (job_desc ilike '%soft%'
  as sw, TRUE as next_job)
  Result (
  First (emp_id of sw) as emp_id,
  Accumulate(job_desc of
  any(sw,next_job)) as Matches)
  ) AS dt
GROUP BY emp_id, Matches
ORDER BY emp_id;
```

- Here, we are specifying the **^** predicate in our **Pattern** argument
- Given this, we are searching for any job-path pattern that *begins the partition* with '%soft%', and then goes to anything else, whatever it may be
- Only **emp\_id 1** has the first row of its partition beginning with '%soft%', thus the job path of **emp\_id 1** is returned, but not the job path of **emp\_id 2** (which begins with 'Janitor')

### Input

	emp id	job desc	bgn dt	end dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31

### Output

	emp id	matches	Count(*)
1	1	[Software Engineer, Director of Engineering]	1



# Symbol Expressions

**Query:** For users whose first title in the partition did not start out like '%Soft%', what was their next title? Also, count the number of times that that path occurred

```
SELECT emp_id, Matches, count (*)
FROM nPath
(ON jobs
PARTITION BY emp_id
ORDER BY bgn_dt
USING
Mode (NONOVERLAPPING)
Pattern ('^sw.next_job')
Symbols (job_desc NOT LIKE '%Soft%' AS sw,
TRUE AS next_job)
Result (First (emp_id of sw) AS emp_id,
Accumulate(job_desc of any(sw,next_job))
AS Matches)
) AS dt
GROUP BY emp_id, Matches
ORDER BY emp_id;
```

## Predicates

### True

symbol can match any row  
(often for exploratory queries)

### % \_

% (any # of char) \_ (positional):  
char comparisons

### like not like

case sensitive text comparison

### ilike not ilike

case insensitive text comparisons

### LAG/LEAD

to compare current row to preceding row(s)

### =, <, >, <=, >=, <>

numeric comparisons

## Input

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31

## Output

	emp_id	matches	Count(*)
1	2	[Janitor, Software Engineer]	1



# Lab 12 – like versus ilike on ML Engine (Optional)

## like

```
SELECT emp_id, Matches, count (*) FROM nPath@coprocessor
(ON jobs PARTITION BY emp_id ORDER BY bgn_dt USING Mode
(NONOVERLAPPING)
Pattern ('sw.next_job')
Symbols (job_desc like 'soft%' as sw, TRUE as next_job)
Result (First (emp_id of sw) as emp_id,
Accumulate(job_desc of any(sw,next_job)) as Matches
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

## ilike

```
SELECT emp_id, Matches, count (*) FROM nPath@coprocessor
(ON jobs PARTITION BY emp_id ORDER BY bgn_dt USING Mode
(NONOVERLAPPING)
Pattern ('^sw.next_job')
Symbols (job_desc ilike 'soft%' as sw, TRUE as next_job)
Result (First (emp_id of sw) as emp_id,
Accumulate(job_desc of any(sw,next_job)) as Matches
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

## Input

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31

## Output for 'like'

	emp_id	matches	Count(*)

## Output for 'ilike'

	emp_id	matches	Count(*)
1	1	[Software Engineer, Director of Engineering]	1
2	2	[Software Engineer, Director of Engineering]	1

- On the ML Engine, the **like** symbol is case-sensitive, whereas the **ilike** symbol is case-insensitive
- Note 'Software Engineer' begins with a capital 'S'
- **like 'soft%'** will return nothing, whereas **ilike 'soft%'** will return rows



## Lab 13 – Using the + Predicate

- Recall the plus symbol ( + ) signifies *occurs at least once (1-n)*
- In the examples below, for each **emp\_id** after a match on **sw**, the **Pattern ('sw.next\_job')** will return only the very next job, whereas the **Pattern ('sw.next\_job+')** will return all subsequent jobs

### Input

	emp_id	job_desc	bgn_dt	end_dt
1	1	Software Engineer	2010-01-01	2010-12-31
2	1	Director of Engineering	2011-01-01	2011-12-31
3	1	CTO	2012-01-01	2012-12-31
4	2	Janitor	2013-01-01	2013-12-31
5	2	Software Engineer	2014-01-01	2014-12-31
6	2	Director of Engineering	2015-01-01	2015-12-31
7	2	CTO	2016-01-01	2016-12-31

### Without +

```
SELECT emp_id, Matches, count (*) FROM  
nPath@coprocessor (ON jobs PARTITION BY emp_id  
ORDER BY bgn_dt USING Mode (NONOVERLAPPING)  
Pattern ('sw.next_job')  
Symbols (job_desc ilike 'soft%' as sw, TRUE as  
next_job)  
Result (First (emp_id of sw) as emp_id,  
Accumulate(job_desc of any(sw,next_job)) as Matches  
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

### With +

```
SELECT emp_id, Matches, count (*) FROM  
nPath@coprocessor (ON jobs PARTITION BY emp_id  
ORDER BY bgn_dt USING Mode (NONOVERLAPPING)  
Pattern ('sw.next_job+')  
Symbols (job_desc ilike 'soft%' as sw, TRUE as  
next_job)  
Result (First (emp_id of sw) as emp_id,  
Accumulate(job_desc of any(sw,next_job)) as Matches  
) AS dt GROUP BY emp_id, Matches ORDER BY emp_id;
```

### Output

	emp_id	matches	Count(*)
1	1	[Software Engineer, Director of Engineering]	1
2	2	[Software Engineer, Director of Engineering]	1

	emp_id	matches		Count(*)
1	1	[Software Engineer, Director of Engineering	CTO]	1
2	2	[Software Engineer, Director of Engineering	CTO]	1



# True Predicate

- The **True** predicate is useful to discover which patterns exist in your data
- Normally, with other predicate values, you are seeking out specific patterns that you explicitly define and know that you are interested in
- **True** is beneficial when you don't know what you're looking for and merely want to discover which patterns may (or may not) exist in your data



## Lab 14 – True Predicate (1 of 2)

- Here, we are familiarizing ourselves with the **bank\_web\_clicks** table
- On the following page, we will use the **TRUE** predicate to discover which web paths are the most-commonly travelled

```
SELECT * FROM bank_web_clicks
WHERE customer_id IN (11603)
ORDER BY customer_id, timestamp;
```

```
CREATE MULTISET TABLE bank_web_clicks ,FALLBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT,
DEFAULT MERGEBLOCKRATIO,
MAP = TD_MAP1
(
customer_id INTEGER,
session_id INTEGER,
page VARCHAR(100) CHARACTER SET LATIN NOT CASESPECIFIC,
timestamp TIMESTAMP(6))
PRIMARY INDEX ( customer_id );
```

	customer id	session id	page	timestamp
1	11603	0	ACCOUNT SUMMARY	2004-04-25 08:37...
2	11603	0	FUNDS TRANSFER	2004-04-25 08:39...
3	11603	0	CUSTOMER SUPPORT	2004-04-25 08:41...
4	11603	0	ACCOUNT SUMMARY	2004-04-25 08:42...
5	11603	0	ACCOUNT SUMMARY	2004-04-25 08:43...
6	11603	0	FUNDS TRANSFER	2004-04-25 08:45...
7	11603	0	ACCOUNT HISTORY	2004-04-25 08:49...
8	11603	0	FAQ	2004-04-25 08:51...
9	11603	0	FUNDS TRANSFER	2004-04-25 08:55...
10	11603	0	ACCOUNT SUMMARY	2004-04-25 08:56...



## Lab 14 – True Predicate (2 of 2)

### Input

```
SELECT path, count(*) AS occurs
FROM nPath
(ON bank_web_clicks
PARTITION BY customer_id, session_id
ORDER BY timestamp
USING
Mode (NONOVERLAPPING)
Pattern ('PAGE+')
Symbols (TRUE AS PAGE)
Result (
Accumulate(page OF ANY (PAGE)) as path)
) AS dt
GROUP BY path
HAVING occurs >= 500
ORDER BY occurs DESC;
```

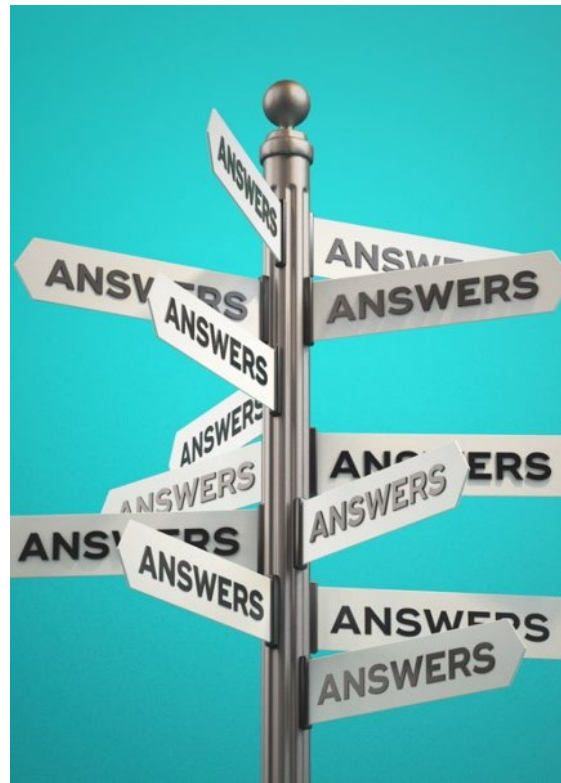
'true as <alias>' means 'the next row'

### Output

	path	occurs
1	[ACCOUNT SUMMARY, FAQ]	2854
2	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS]	2830
3	[ACCOUNT SUMMARY, FUNDS TRANSFER]	2746
4	[ACCOUNT SUMMARY, ACCOUNT SUMMARY]	2715
5	[ACCOUNT SUMMARY, ACCOUNT HISTORY]	2699
6	[ACCOUNT SUMMARY, ONLINE STATEMENT ENROLLMENT]	698
7	[ACCOUNT SUMMARY, PROFILE UPDATE]	684
8	[ACCOUNT SUMMARY, CUSTOMER SUPPORT]	647
9	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	575
10	[ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT SUMMARY]	571
11	[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY]	549
12	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]	545
13	[ACCOUNT SUMMARY, VIEW DEPOSIT DETAILS, FUNDS TRANSFER]	524
14	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, ACCOUNT HISTORY]	510
15	[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS]	507
16	[ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY]	502

# Current Topic – Results

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- Symbols
- Mode
- Pattern and Symbols
- **Results**
- Teradata AppCenter
- Hackathon and Review





## Result (1 of 2)

- **RESULT** is the output for each matched **PATTERN** in the sequence of rows
- **nPath** generates a row of output that can contain SQL and/or ML aggregates computed over the rows within the matched **PATTERN**

**SYNTAX** `Result ({ aggregate_function (col_expr OF symbol) AS alias_1 }[,...])`

```
SELECT emp_id, Matches, count (*)
FROM nPath@coprocessor
(ON jobs PARTITION BY emp_id ORDER BY bgn_dt
USING
Mode (NONOVERLAPPING)
Pattern ('sw.next_job')
Symbols (job_desc ilike '%soft%' AS sw, TRUE AS next_job)
Result (First (emp_id OF sw) AS emp_id,
        Accumulate(job_desc OF ANY(sw,next_job)) AS Matches)
) AS dt
GROUP BY emp_id, Matches
ORDER BY emp_id;
```

- Here, we are using **First** and **Accumulate** in our **Result** argument
- **First** returns the `col_expr` value of the first matched row
- **Accumulate** returns, for each matched row, the concatenated values in `col_expr`, separated by a delimiter. The default delimiter is a comma followed by a blank space ( , )



## Result (2 of 2)

Following are some common Aggregates that you can specify in the Result argument:

- **COUNT**(\* of SYMBOL)
- **FIRST**(<expr> of SYMBOL)
- **LAST**(<expr> of SYMBOL)
- **SUM**(<expr> of SYMBOL)
- **AVG**(<expr> of SYMBOL)
- **MAX**(<expr> of SYMBOL)
- **MIN**(<expr> of SYMBOL)
- **ACCUMULATE**(<expression> of SYMBOL)
- **ANY**: e.g., SUM(<expression> of ANY(A,B,C))
- **FIRST\_NOTNULL** (column of symbol): Returns first non-null row that maps to symbol
- **LAST\_NOTNULL** (column of symbol): Returns last non-null row that maps to symbol
- **MAX\_CHOOSE** (qty column, column name): Returns descriptive column of highest qty col
- **MIN\_CHOOSE** (qty column, column name): Returns descriptive column of lowest qty col
- **DUPCOUNT**: Counts # of times value has appeared preceding this row
- **DUPCOUNTCUM**: # of Duplicate values have appeared contiguously preceding



## Lab 15 – Result Arguments (1 of 4)

- Over the next many pages, we will walk through various **Result** arguments using the **web\_purchases** table, shown below
- Note:
  - There are three **user\_id** values
  - user\_id 1** had two sessions. The other two **user\_id** values had only one **session\_id** each
  - user\_id 1** bought a *guitar*, **user\_id 2** bought *strings* as well as a *guitar*, and **user\_id 3** bought a *capo*

```
SELECT * FROM web_purchases  
ORDER BY user_id, date_time;
```

	user_id	session_id	date_time	page	product	prod qty	prod price
1	1	0	2018-01-01 13:21...	home	null	null	null
2	1	0	2018-01-01 13:22...	contact_us	null	null	null
3	1	0	2018-01-01 13:23...	view_prod	guitar	1	null
4	1	0	2018-01-01 13:24...	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21...	home	null	null	null
6	1	1	2018-01-02 14:22...	about us	null	null	null
7	2	0	2018-02-01 13:21...	home	null	null	null
8	2	0	2018-02-01 13:22...	view_prod	strings	null	null
9	2	0	2018-02-01 13:23...	view_prod	guitar	null	null
10	2	0	2018-02-01 13:24...	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25...	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22...	home	null	null	null
13	3	0	2018-04-01 14:23...	view_prod	capo	null	null
14	3	0	2018-04-01 14:24...	checkout	capo	1	15.00



## Lab 15 – Result Arguments (2 of 4)

```
SELECT * FROM nPath
(ON web_purchases
PARTITION BY user_id, session_id
ORDER BY date_time USING
Mode (NONOVERLAPPING) Pattern ('V+.C+')
Symbols (page = 'view_prod' as V, page = 'checkout' as C)
Result (Accumulate(product of any(V, C)) as Matches)
) AS dt;
```

### Input

	user_id	session_id	date_time	page	product	prod_atv	prod_price
1	1	0	2018-01-01 13:21...	home	null	null	null
2	1	0	2018-01-01 13:22...	contact_us	null	null	null
3	1	0	2018-01-01 13:23...	view_prod	guitar	1	null
4	1	0	2018-01-01 13:24...	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21...	home	null	null	null
6	1	1	2018-01-02 14:22...	about_us	null	null	null
7	2	0	2018-02-01 13:21...	home	null	null	null
8	2	0	2018-02-01 13:22...	view_prod	strings	null	null
9	2	0	2018-02-01 13:23...	view_prod	guitar	null	null
10	2	0	2018-02-01 13:24...	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25...	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22...	home	null	null	null
13	3	0	2018-04-01 14:23...	view_prod	capo	null	null
14	3	0	2018-04-01 14:24...	checkout	capo	1	15.00

- In the example, we are searching within each partition for a **Pattern** of *one or more instances of view\_prod, followed by one or more instances of checkout*
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern** within each partition

### Output

	matches
1	[guitar, guitar]
2	[strings, guitar, strings, guitar]
3	[capo, capo]



## Lab 15 – Result Arguments (3 of 4)

```
SELECT * FROM nPath (ON web_purchases
PARTITION BY user_id, session_id ORDER BY date_time
USING Mode (NONOVERLAPPING) Pattern ('V+.C+'))
Symbols (page = 'view_prod' as V, page='checkout' as C)
Result (
  First (user_id of V) as user_id,
  First (session_id of V) as session_id,
  Accumulate (product of any(V, C)) as Matches)
) AS dt ORDER BY user_id, session_id;
```

- Our **First** arguments return the very first instance of **user\_id** that viewed the specified product within the partition
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern** within the partition

### Input

	user id	session id	date time	page	product	prod qty	prod price
1	1	0	2018-01-01 13:21...	home	null	null	null
2	1	0	2018-01-01 13:22...	contact_us	null	null	null
3	1	0	2018-01-01 13:23...	view_prod	guitar	1	null
4	1	0	2018-01-01 13:24...	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21...	home	null	null	null
6	1	1	2018-01-02 14:22...	about_us	null	null	null
7	2	0	2018-02-01 13:21...	home	null	null	null
8	2	0	2018-02-01 13:22...	view_prod	strings	null	null
9	2	0	2018-02-01 13:23...	view_prod	guitar	null	null
10	2	0	2018-02-01 13:24...	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25...	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22...	home	null	null	null
13	3	0	2018-04-01 14:23...	view_prod	capo	null	null
14	3	0	2018-04-01 14:24...	checkout	capo	1	15.00

### Output

	user id	session id	matches
1	1	0	[guitar, guitar]
2	2	0	[strings, guitar, strings, guitar]
3	3	0	[capo, capo]



## Lab 15 – Result Arguments (4 of 4)

```
SELECT * FROM nPath
(ON web_purchases
PARTITION BY user_id, session_id
ORDER BY date_time
USING
Mode (NONOVERLAPPING) Pattern ('V+.C+')
Symbols
(page = 'view_prod' AS V, page='checkout' AS C)
Result (
  First (user_id of V) AS user_id,
  First (session_id of V) AS session_id,
  Accumulate (product of any(V, C)) AS Matches,
  Count (distinct product of any(C)) AS cd_Matches,
  Sum (cast(prod_price as integer) of any (C)) AS
    total_price)
) AS dt ORDER BY user_id, session_id;
```

### Input

	user_id	session_id	date_time	page	product	prod_atv	prod_price
1	1	0	2018-01-01 13:21...	home	null	null	null
2	1	0	2018-01-01 13:22...	contact_us	null	null	null
3	1	0	2018-01-01 13:23...	view_prod	guitar	1	null
4	1	0	2018-01-01 13:24...	checkout	guitar	1	250.00
5	1	1	2018-01-02 14:21...	home	null	null	null
6	1	1	2018-01-02 14:22...	about_us	null	null	null
7	2	0	2018-02-01 13:21...	home	null	null	null
8	2	0	2018-02-01 13:22...	view_prod	strings	null	null
9	2	0	2018-02-01 13:23...	view_prod	guitar	null	null
10	2	0	2018-02-01 13:24...	checkout	strings	5	25.00
11	2	0	2018-02-01 14:25...	checkout	guitar	1	1000.00
12	3	0	2018-04-01 14:22...	home	null	null	null
13	3	0	2018-04-01 14:23...	view_prod	capo	null	null
14	3	0	2018-04-01 14:24...	checkout	capo	1	15.00

- Our **Sum** argument sums the **prod\_price** of qualifying *checkout* products
- Our **Count** argument counts the distinct products that were present in a *checkout*
- Our **First** arguments return the very first instance of **user\_id** that viewed the specified product
- Our **Accumulate** argument returns any instances of product that meet the conditions of our **Pattern**

### Output

	user_id	session_id	matches	cd_matches	total price
1	1	0	[guitar, guitar]	1	250
2	2	0	[strings, guitar, strings, guitar]	2	1025
3	3	0	[capo, capo]	1	15

# Current Topic – Teradata AppCenter

- Background Information
  - Description
  - Use Cases
  - Syntax
  - Input Data
  - Required Arguments
  - Optional Arguments
  - Input Table Schema
- Symbols
- Mode
- Pattern and Symbols
- Results
- **Teradata AppCenter**
- Hackathon and Review



## Current Topic

- Using R Studio with Teradata Vantage
  - Teradata Vantage architecture
  - Install and load dependent packages
  - Connect to Teradata Vantage from R Studio
  - What is a tibble?
- Transformation Functions
  - Scale and scale map, Text parser
- Association Analysis
  - Collaborative filtering
- **Path and Pattern Analysis**
  - **nPath**
- Statistical Functions
  - Decision forests, K-means



## nPath Description (1 of 2)

- The **nPath** function scans a set of rows, looking for patterns that you specify
- For each set of input rows that matches the pattern, **nPath** produces a single output row
- The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set
- **nPath** is useful when your goal is to identify the paths that lead to an outcome
- The output from the **nPath** function can be input into other Machine Learning Engine functions or into a visualization tool such as Teradata AppCenter

The **td\_nPath** function scans a set of rows, looking for patterns that you specify. For each set of input rows that matches the pattern, **td\_nPath** produces a single output row. The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.

## nPath Description (2 of 2)

### What is nPath?

- Function designed for time-series sequence analysis of data
- Links an outcome with a preceding path

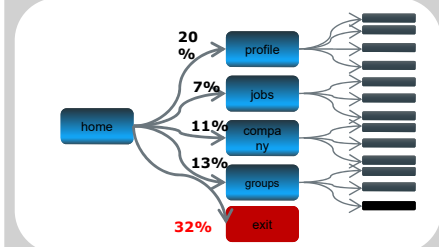
### Benefits

- *Pattern detection can be completed in a single pass over the data*
- Allows you to understand relationships across rows of data
- Transcends SQL's ordered-data limitations that require either complex, multi-pass SQL or custom UDFs for each analysis

### Example use cases:

- Web analytics (clickstream, Golden Path)
- Complex Marketing revenue paths
- Granular product & process analysis (A/B)
- Granular pattern detection (fraud, QA,..)

### nPath Website Path Analysis



### Complete Application:

- **nPath** identifies path patterns and exit points
- The **Sessionize** function is used prepare the input data for **nPath** analysis.
- In the example above, we are viewing which paths users take once they've landed on our Home page. Note that 32% of Home page visits end up in the users' exiting the website altogether.

## nPath Use Cases

Some examples of how **nPath** can be used follow:

- A **retailer** wishes to analyze Web site click data, to identify paths that lead to sales over a specified amount
- A **manufacturer** analyzes sensor data from industrial processes, to identify paths to poor product quality
- A **healthcare provider** analyzes healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- A **financial institution** reviews financial data for individuals, to identify paths that provide information about credit or fraud risks

nPath is useful when your goal is to identify the paths that lead to an outcome. For example, you can use nPath to analyze:

- Web site click data, to identify paths that lead to sales over a specified amount
- Sensor data from industrial processes, to identify paths to poor product quality
- Healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- Financial data for individuals, to identify paths that provide information about credit or fraud risks

The output from the nPath function can be input to other ML Engine functions or to a visualization tool such as Teradata AppCenter.

## nPath Workflow



- **Input Tibble:** Data is read from specified input tables
- **nPath:** The following arguments are specified when the function is invoked
  - **Mode (overlapping or nonoverlapping)**
  - **Pattern to match**
  - **Symbols to use**
  - **[Optional] Filters to apply**
  - **Results to output**
- **Output object:** Data is written to an output object

**nPath** requires at least one input table, view, or query. Rows that meet the condition of your logic are then displayed in the output.



## Lab 10 – Bb\_borre\_z Remote Tibble

```
bb_borre_z <-tbl(con,  
dplyr::sql ("SELECT * FROM bb_borre_z"))
```

Create a remote tibble named  
**bb\_borre\_z**

1. Use the **tbl** function
2. Reference our **con** Vantage context variable
3. Use the **dplyr::sql** function to query the Vantage table

	user_id	event	ts
	<i>&lt;int&gt;</i>	<i>&lt;chr&gt;</i>	<i>&lt;dtm&gt;</i>
1	1	a	2017-01-01 13:21:03
2	1	a	2017-01-01 13:21:04
3	1	a	2017-01-01 13:21:02
4	1	a	2017-01-01 13:21:01

Input Tibble

## nPath – Syntax

- The `td_nPath_sql` function scans a set of rows, looking for patterns that you specify.
- For each set of input rows that matches the pattern, `nPath` produces a single output row.
- The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.

```
td_npath_sql (
  data1 = NULL,
  mode = NULL,
  pattern = NULL,
  symbols = NULL,
  result = NULL,
  filter = NULL,
  data2 = NULL,
  data3 = NULL,
  data1.partition.column = NULL,
  data2.partition.column = NULL,
  data3.partition.column = NULL,
  data1.order.column = NULL,
  data2.order.column = NULL,
  data3.order.column = NULL)
```

Following are important points to realize about the syntax for `nPath`.

The *required* arguments for `nPath` follow:

- **Mode**: Specify the pattern-matching mode. Possible values include **OVERLAPPING** and **NONOVERLAPPING**.
- **Pattern**: Specify the pattern for which the function searches.
- **Symbols**: Specify the symbols that appear in the values of the **Pattern** and **Result** arguments.
- **Result**: Defines the output columns.

The *optional* arguments for `nPath` follow:

- **Filter** [Optional]: Specify filters to impose on the matched rows. The function combines the filter expressions using the AND operator.

## nPath Required Arguments: Mode

Here, we specify the pattern-matching mode. There are two flavors of this:

- **OVERLAPPING**: Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern
- **NONOVERLAPPING**: Start next pattern search at row that follows last pattern match

Mode allows you to specify the pattern-matching mode. The two options follow:

- **OVERLAPPING**: Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern.
- **NONOVERLAPPING**: Start next pattern search at row that follows last pattern match.

## nPath Required Arguments: Pattern

Here, we specify the **Pattern** for which the function searches.

- We compose **Pattern** with symbols (which we define in the **Symbols** argument), operators, and parentheses. Here, we are searching for **Symbol X followed by X** (which means event 'a' followed by event 'a')
- When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses

**Pattern** allows you to specify the pattern for which the function searches. You compose *pattern* with the symbols (which you define in the **Symbols** argument), operators, and parentheses.

When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses.

## Pattern Operators (1 of 2)

- Use with pattern symbols to customize pattern-matching rules
  - '.' : followed by (Use to separate a series of pattern symbols)
  - '|' : alternative (The equivalent of an OR)
  - '?' : occurs at most once (0-1)
  - '\*' : occurs zero or more times (0-n)
  - '+' : occurs at least once (1-n)
  - '^' : pattern must begin with value specified. Also, value specified must be the first row within the partition.
  - '\$' : pattern must end with
- Customizing pattern matching rules:
  - (X){a}: exactly A number of occurrences of X `pattern('A.B{3}')`
  - (X){a,}: at least A number of occurrences of X `pattern('A.B{1,}')`
  - (X){a,b}: A to B occurrences of X `pattern('A.B{1,3}')`

The following pages show various operators that you can use when defining your nPath patterns.

## Pattern Operators (2 of 2)

Operator	Description	Precedence
<i>A</i>	Matches one row that meets the definition of A	1 (highest)
<i>A.</i>	Matches one row that meets the definition of A	1
<i>A?</i>	Matches 0 or 1 rows that satisfy the definition of A	1
<i>A*</i>	Matches 0 or more rows that satisfy the definition of A (greedy operator)	1
<i>A+</i>	Matches 1 or more rows that satisfy the definition of A (greedy operator)	1
<i>A.B</i>	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B	2
<i>A B</i>	Matches one row that meets the definition of either A or B	3

The **nPath** function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators

## nPath Required Argument – Symbols

Here, we specify the **Symbols** that appear in the values of the **Pattern** and **Result** parameters

- The *col\_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol\_predicate* is a SQL predicate (often a column name)
- You can think of **Symbols** as the “aliases” that you will be defining for use in the **Pattern** and **Result** portions of the query
- The symbol is **case-insensitive**; however, a symbol of one or two uppercase letters is easy to identify in patterns. If *col\_expr* represents a column that appears in multiple input tables, then you must qualify the ambiguous column name with its table name.

```
pattern = "X.X",
symbols = c("event = 'a' as X"),
```

Symbols allows you to define the symbols that appear in the values of the Pattern and Result arguments. The *col\_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol\_predicate* is a predicate (often a column name).

For example, this Symbols argument is for analyzing website visits:

```
Symbols (
  pagetype = 'homepage' AS H,
  pagetype <> 'homepage' AND pagetype <> 'checkout' AS PP,
  pagetype = 'checkout' AS CO
)
```

The *symbol* is case-insensitive; however, a *symbol* of one or two uppercase letters is easy to identify in patterns.

If *col\_expr* represents a column that appears in multiple input tables, you must qualify the ambiguous column name with its table name. For example:

```
Symbols (
  weblog.pagetype = 'homepage' AS H,
  weblog.pagetype = 'thankyou' AS T,
  ads.adname = 'xmaspromo' AS X,
  ads.adname = 'realtorpromo' AS R
)
```

## nPath Required Arguments: Result

Here, we specify the output columns

- The **col\_expr** is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies **aggregate\_function** to these values
- The function evaluates this argument once for every matched pattern in the partition (that is, it outputs one row for each pattern match)

```
pattern = "X.X",
symbols = c("event = 'a' as X"),
result = c("ACCUMULATE (event of X) AS x_pattern")
```

	x_pattern <chr>
1	[a, a]
2	[a, a]

**Result** allows you to define the output columns.

The *col\_expr* is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies *aggregate\_function* to these values. The function evaluates this argument once for every matched pattern in the partition (that is, it outputs one row for each pattern match).



## Lab 11 – Simple nPath Example

```
np_path_out <- td_npath_sql(
  data1 = bb_borre_z,
  data1.partition.column = c("user_id"),
  data1.order.column = "ts",
  mode = "nonoverlapping",
  pattern = "X.X",
  symbols = c("event = 'a' as X"),
  result = c("ACCUMULATE (event of X) AS x_pattern"))
```

	x_pattern
	<chr>
1	[a, a]
2	[a, a]

Create an object named **bb\_borre\_z**

1. Use the **tbl\_npath\_sql** function
2. Reference our **bb\_borre\_z** remote tibble
3. Select **user\_id** as the partition column
4. Order by the **ts** column
5. Input the remaining required arguments

- On the following pages, we will discuss the required arguments for **nPath**
- For each required argument, we will discuss the implications of our specifications using the simple **nPath** query to the right as the foundation



## Lab 12 – nPath Between Remote Tibble

```
npath_between <-tbl(con,  
dplyr::sql ("SELECT * FROM bb_npathBetween2"))
```

Create a remote tibble named  
**npath\_between**

1. Use the **tbl** function
2. Reference our **con** Vantage context variable
3. Use the **dplyr::sql** function to query the Vantage table

	c1	c2	c3
	<int>	<int>	<chr>
1	1	1	B
2	3	1	C
3	2	1	B
4	5	1	D
5	4	1	A



## Lab 13a – nPath 'Or' Pattern

```
npath_or_out <-td_npath_sqlc(  
  data1 = npath_between,  
  data1.partition.column = c("c1"),  
  data1.order-column = "c1",  
  mode = "nonoverlapping",  
  pattern = "B|C|A",  
  symbols = c("c3 = 'A' as A", "c3 = 'B' as B",  
              "c3 = 'C' as C")  
  results = c ("ACCUMULATE(c3 of ANY(B,C,A))  
              AS matches"))
```

Following our regex and delimitating our pattern with | symbols, we are telling our nPath code that we want B or C or A as the output.



## Lab 13b – Output

View the output by typing the following command into the console:

```
npath_or_out
```

### Input

	c1	c2	c3
	<i>&lt;int&gt;</i>	<i>&lt;int&gt;</i>	<i>&lt;chr&gt;</i>
1	1	1	B
2	3	1	C
3	2	1	B
4	5	1	D
5	4	1	A

### Output

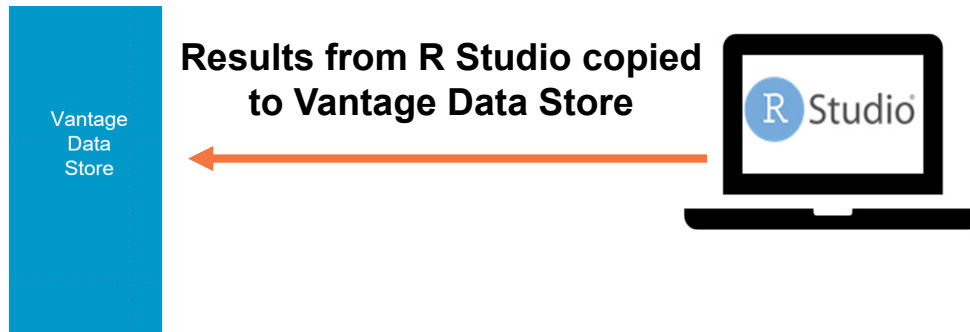
	matches
	<i>&lt;chr&gt;</i>
1	[B]
2	[C]
3	[B]
4	[A]

Again we can use dplyr to arrange the token by the most-frequently used tokens.

## Move Results from R Studio to a Vantage Table

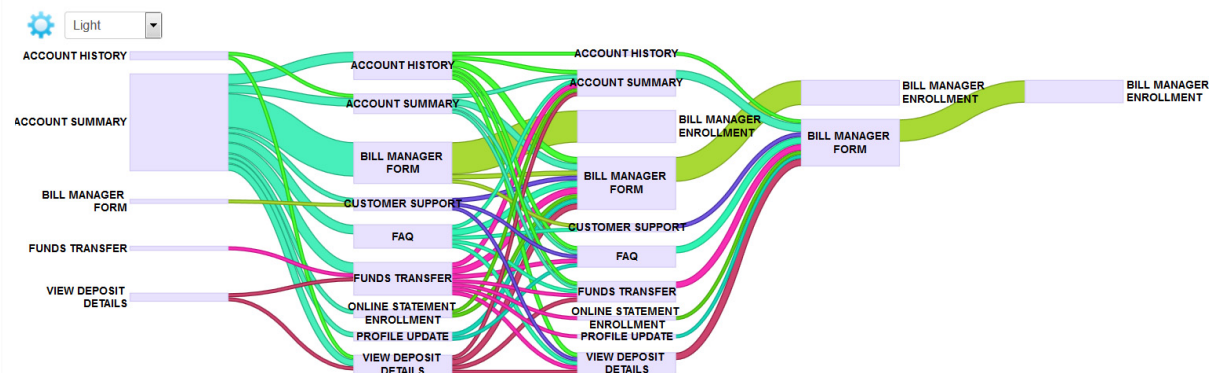
To more easily create visualization in **Teradata AppCenter** we can **copy** our results from **R Studio** into a **Vantage table**

```
copy_to(con,npath_or_out$result,name = "npath_or_out")
```



## Teradata AppCenter Example

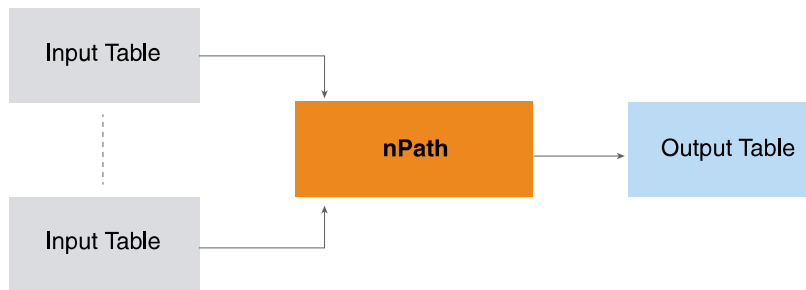
- Teradata AppCenter is a Web-based application that allows us to build visualization applications to be able to view data in various types of chart displays
- It is especially useful for viewing the results of **nPath** queries
- In the example below, we are viewing the results of an **nPath** query in a Sankey chart to discover the most common paths that lead towards **Bill Manager Enrollment**



# nPath® (ML Engine)

## Introduction to nPath

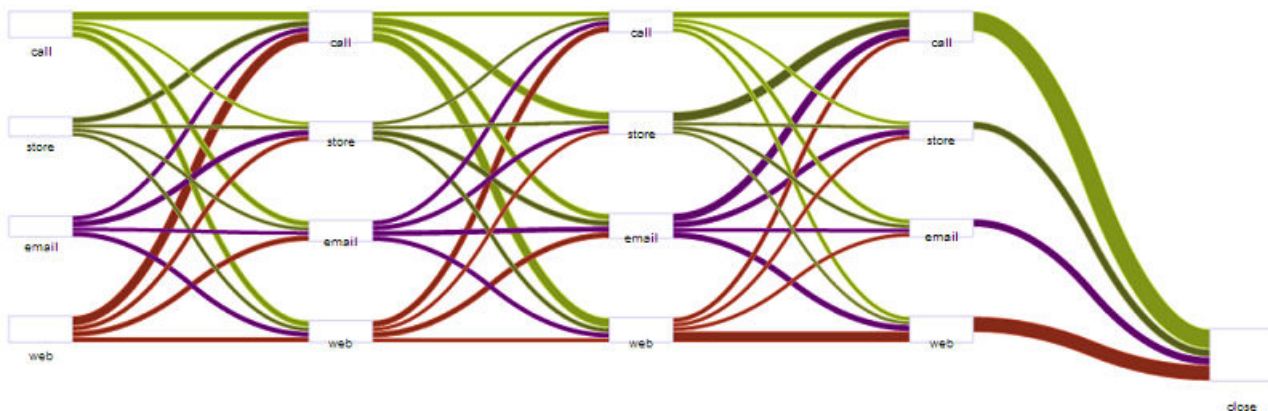
The nPath function scans a set of rows, looking for patterns that you specify. For each set of input rows that matches the pattern, nPath produces a single output row. The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.



nPath is useful when your goal is to identify the paths that lead to an outcome. For example, you can use nPath to analyze:

- Web site click data, to identify paths that lead to sales over a specified amount
- Sensor data from industrial processes, to identify paths to poor product quality
- Healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- Financial data for individuals, to identify paths that provide information about credit or fraud risks

The output from the nPath function can be input to other ML Engine functions or to a visualization tool such as Teradata® AppCenter.

**Sankey Diagram of ML Engine nPath Output**

An nPath call specifies:

- [Mode \(overlapping or nonoverlapping\)](#)
- [Pattern to match](#)
- [Symbols to use](#)
- [Optional] [Filters to apply](#)
- [Results to output](#)

## nPath Syntax

### Version 1.1

#### Note:

This function requires "@coprocessor".

```
SELECT * FROM nPath@coprocessor (
  ON { table | view | (query) } PARTITION BY partition_column ORDER BY order_column [ ASC
  | DESC ][...]
  [ ON { table | view | (query) }
    [ PARTITION BY partition_column | DIMENSION ] ORDER BY order_column [ ASC | DESC ]
  ][...]
  USING
  Mode ({ OVERLAPPING | NONOVERLAPPING })
  Pattern ('pattern')
  Symbols ({ col_expr = symbol_predicate AS symbol } [,...])
  [ Filter (filter_expression [,...]) ]
  Result ({ aggregate_function (col_expr OF symbol) AS alias_1 } [,...])
) AS alias_2;
```

The nPath function is not tied to any schema and must not be qualified with a schema name.

#### Related Information:

[Comments in Queries](#)

## nPath Syntax Elements

### Mode

Specify the pattern-matching mode:

Option	Description
OVERLAPPING	Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern.
NONOVERLAPPING	Start next pattern search at row that follows last pattern match.

### Pattern

Specify the pattern for which the function searches. You compose *pattern* with the symbols (which you define in the Symbols syntax element), operators, and parentheses.

When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses. For more information, see [nPath Patterns](#).

### Symbols

Specify the symbols that appear in the values of the Pattern and Result syntax elements. The *col\_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol\_predicate* is a SQL predicate (often a column name).

For example, this Symbols syntax element is for analyzing website visits:

```
Symbols (
  pagetype = 'homepage' AS H,
  pagetype <> 'homepage' AND pagetype <> 'checkout' AS PP,
  pagetype = 'checkout' AS CO
)
```

The *symbol* is case-insensitive; however, a *symbol* of one or two uppercase letters is easy to identify in patterns.

If *col\_expr* represents a column that appears in multiple input tables, you must qualify the ambiguous column name with its table name. For example:

```
Symbols (
  weblog.pagetype = 'homepage' AS H,
  weblog.pagetype = 'thankyou' AS T,
  ads.adname = 'xmaspromo' AS X,
```

```
ads.adname = 'realtorpromo' AS R
)
```

For more information about symbols that appear in the Pattern syntax element value, see [nPath Symbols](#). For more information about symbols that appear in the Result syntax element value, see [nPath Results](#).

## Filter

[Optional] Specify filters to impose on the matched rows. The function combines the filter expressions using the AND operator.

This is the *filter\_expression* syntax:

```
symbol_expression comparison_operator symbol_expression
```

The two symbol expressions must be type-compatible. This is the *symbol\_expression* syntax:

```
{ FIRST | LAST }(column_with_expression OF [ANY](symbol[, ...]))
```

The *column\_with\_expression* cannot contain the operator AND or OR, and all its columns must come from the same input. If the function has multiple inputs, *column\_with\_expression* and *symbol* must come from the same input.

The *comparison\_operator* is either <, >, <=, >=, =, or !=.

Whether this syntax element improves or degrades nPath performance depends on several factors. For details, see [nPath Filters](#).

## Result

Specify the output columns. The *col\_expr* is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies *aggregate\_function* to these values. For details, see [nPath Results](#).

The function evaluates this syntax element once for every matched pattern in the partition (that is, it outputs one row for each pattern match).

# nPath Input

The function requires at least one partitioned input table, and can have additional input tables that are either partitioned or DIMENSION tables.

## Note:

If the input to nPath is nondeterministic, the results are nondeterministic. For more information, see [Nondeterministic Results and UniqueID Syntax Element](#).

## Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	INTEGER or VARCHAR	Column by which every partitioned input table is partitioned.
<i>order_column</i>	INTEGER or VARCHAR	Column by which every input table is ordered.
<i>input_column</i>	INTEGER or VARCHAR	Data to search for patterns.

## nPath Output

### Output Table Schema

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	Column by which partitioned input tables are partitioned.
<i>order_column</i>	Same as in input table	Column by which input tables are ordered.
<i>result_column</i>	Same as result of <i>aggregate_function</i>	Determined by Result syntax element. For details, see <a href="#">nPath Results</a> .

## nPath Symbols

A *symbol* identifies a row in the Pattern and Result syntax elements. A symbol can be any valid identifier (that is, a sequence of characters and digits that begins with a character) but is typically one or two uppercase letters. Symbols are case-insensitive; that is, 'SU' is identical to 'su', and the system reports an error if you use both.

For example, suppose that you have this input table:

record	city	temp	rh	cloudcover	windspeed	winddirection	rained_next_day
1	Tucson	81	30	0.0	5	NW	1
2	Tempe	76	40	0.2	15	NE	0
3	Tucson	70	70	0.4	10	N	0
4	Tusayan	75	50	0.4	5	NW	0

This table has examples of symbol definitions and the rows of the table that they match in NONOVERLAPPING mode:

Symbol Definition	Rows Matched
temp >= 80 AS H	1

Symbol Definition	Rows Matched
<code>winddirection = 'NW' AS NW</code>	1, 4
<code>winddirection = 'NW' OR windspeed &gt; 12 AS W</code>	1, 2, 4
<code>cloudcover != 0.0 AND rh &gt; 35 AS C</code>	2, 3, 4 (An alternative to != is <>.)
<code>'true' AS A</code>	1, 2, 3, 4 This symbol definition matches all rows, for any input table.
<code>city like 'tu%' AS TU</code>	None The like operator is case-sensitive. The % operator matches any number of characters.
<code>city not like 'tu%' AS TU</code>	None
<code>city ilike 'tu%' AS TU</code>	1, 3, 4 The ilike operator is case-insensitive.
<code>city not ilike 'tu%' AS N</code>	2
<code>city ilike 'tu%n' as T</code>	1, 3, 4 The % operator matches any number of characters.
<code>city ilike 'tu__n' as T</code>	1, 3 The underscore (_) operator matches any single character. The pattern 'tu__n' has three underscores, so it matches 'Tucson' but not 'Tusayan'.

Rows with NULL values do not match any symbol. That is, the function ignores rows with missing values.

## LAG and LEAD Expressions in Symbol Predicates

You can create symbol predicates that compare a row to a previous or subsequent row, using a LAG or LEAD operator.

### LAG Expression Syntax

```
{ current_expr operator LAG (previous_expr, lag_rows [, default]) |
  LAG (previous_expr, lag_rows [, default]) operator current_expr }
```

#### **current\_expr**

Name of a column from the current row, or an expression operating on a column from the current row.

#### **operator**

Either >, >=, <, <=, =, or !=.

***previous\_expr***

Name of a column from a previous row, or an expression operating on a column from a previous row.

***lag\_rows***

Number of rows to count backward from the current row to reach the previous row. For example, if *lag\_rows* is 1, the previous row is the immediately preceding row.

***default***

Value to use for *previous\_expr* when there is no previous row (that is, when the current row is the first row or there is no row that is *lag\_rows* before the current row).

**LAG and LEAD Expression Rules**

- A symbol definition can have multiple LAG and LEAD expressions.
- A symbol definition that has a LAG or LEAD expression cannot have an OR operator.
- If a symbol definition has a LAG or LEAD expression and the input is not a table, you must create an alias of the input query, as in [LAG and LEAD Expressions Example: Input Query with Alias](#).

**LAG and LEAD Expressions Example: Input Query with Alias****Input****bank\_web\_clicks**

customer_id	session_id	page	timestamp
529	0	ACCOUNT SUMMARY	2004-03-17 16:35:00
529	0	FAQ	2004-03-17 16:38:00
529	0	ACCOUNT HISTORY	2004-03-17 16:42:00
529	0	FUNDS TRANSFER	2004-03-17 16:45:00
529	0	ONLINE STATEMENT ENROLLMENT	2004-03-17 16:49:00
529	0	PROFILE UPDATE	2004-03-17 16:50:00
529	0	ACCOUNT SUMMARY	2004-03-17 16:51:00
529	0	CUSTOMER SUPPORT	2004-03-17 16:53:00
529	0	VIEW DEPOSIT DETAILS	2004-03-17 16:57:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:16:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:18:00
529	1	FAQ	2004-03-18 01:20:00
...	...	...	...

**SQL Call**

```

SELECT * FROM nPath@coprocessor (
  ON (SELECT customer_id, session_id, datestamp, page FROM bank_web_clicks) AS
  alias1
  PARTITION BY customer_id, session_id
  ORDER BY datestamp
  USING
  Mode (NONOVERLAPPING)
  Pattern ('(DUP|A)*')
  Symbols (
    'true' AS A,
    page = LAG (page,1) AS DUP
  )
  Result (
    FIRST (customer_id OF any (A)) AS customer_id,
    FIRST (session_id OF A) AS session_id,
    FIRST (datestamp OF A) AS first_date,
    LAST (datestamp OF ANY(A,DUP)) AS last_date,
    ACCUMULATE (page OF A) AS page_path,
    ACCUMULATE (page of DUP) AS dup_path)
) AS dt GROUP BY 1;

```

## Output

customer_id	session_id	first_date	last_date	page_path	dup_path
529	0	2004-03-17 16:35:00	2004-03-17 16:57:00	[ACCOUNT SUMMARY, FAQ, ACCOUNT HISTORY, FUNDS TRANSFER, ONLINE STATEMENT ENROLLMENT, PROFILE UPDATE, ACCOUNT SUMMARY, CUSTOMER SUPPORT, VIEW DEPOSIT DETAILS]	[]
529	1	2004-03-18 01:16:00	2004-03-18 01:28:00	[ACCOUNT SUMMARY, FAQ, ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, ACCOUNT SUMMARY, ACCOUNT HISTORY]	[ACCOUNT SUMMARY]
529	2	2004-03-18 09:22:00	2004-03-18 09:36:00	[ACCOUNT SUMMARY, ACCOUNT HISTORY, FUNDS TRANSFER, ACCOUNT SUMMARY, FAQ]	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]
529	3	2004-03-18 22:41:00	2004-03-18 22:55:00	[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY, ACCOUNT HISTORY, FAQ, ACCOUNT SUMMARY]	[ACCOUNT SUMMARY]
529	4	2004-03-19 08:33:00	2004-03-19 08:41:00	[ACCOUNT SUMMARY, FAQ, VIEW DEPOSIT DETAILS, FAQ]	[]
529	5	2004-03-19 10:06:00	2004-03-19 10:14:00	[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	[VIEW DEPOSIT DETAILS]
...	...	...	...	...	...

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## LAG and LEAD Expressions Example: First and Most Expensive Purchases

Whenever a user visits the home page and then visits checkout pages and buys increasingly expensive products, the nPath query returns the first purchase and the most expensive purchase.

### Input

The input table is a collection of clickstream data for different products with price information. Columns userid and sessionid identify the users.

#### aggregate\_clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company2	300
1039	3	television	home	2009-05-23 13:17:59	Company3	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company4	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company4	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company4	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1040	2	Appliances	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company4	400
1040	4	chairs	home1	2009-07-16 11:18:16	Company4	400
1040	4	chairs	page1	2009-07-16 11:18:18	Company4	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...	...	...	...	...	...	...

## SQL Call

```

SELECT * FROM nPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid
  ORDER BY clicktime ASC, productname, pagetype, userid
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.X*.P1.P2+')
  Symbols (
    'true' AS X,
    pagetype = 'home' AS H,
    pagetype <> 'home' AND pagetype <> 'checkout' AS D,
    pagetype = 'checkout' AS P1,
    pagetype = 'checkout' AND
    productprice > 100 AND
  
```

```

    productprice > LAG (productprice, 1, 100::REAL) AS P2
  )
  Result (
    FIRST (productname OF P1) AS first_product,
    MAX_CHOOSE (productprice, productname OF P2) AS max_product,
    FIRST (sessionid OF P2) AS sessionid
  )
) AS dt;

```

## Output

first_product	max_product	sessionid
bookcases	cellphones	5

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Patterns

The value of the Pattern syntax element specifies the sequence of rows for which the function searches. You compose the pattern definition, *pattern*, with symbols (which you define in the Symbols syntax element), operators, and parentheses. In the pattern definition, symbols represent rows. You can combine symbols with pattern operators to define simple or complex patterns of rows for which to search.

### Basic Pattern Operators

The following table lists and describes the basic pattern operators, in decreasing order of precedence. In the table, A and B are symbols that have been defined in the Symbols syntax element.

Operator	Description	Precedence
A	Matches one row that meets the definition of A.	1 (highest)
A.	Matches one row that meets the definition of A.	1
A?	Matches 0 or 1 rows that satisfy the definition of A.	1
A*	Matches 0 or more rows that satisfy the definition of A (greedy operator).	1
A+	Matches 1 or more rows that satisfy the definition of A (greedy operator).	1
A.B	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B.	2
A B	Matches one row that meets the definition of either A or B.	3

The nPath function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators. For more information, see [nPath Greedy Pattern Matching](#).

## Pattern Operator Precedence

Example	Equivalent
A.B+	A.(B+)
A B*	A (B*)
A.B C	(A.B) C

Example:

```
A.(B|C)+.D?.X*.A
```

The preceding pattern definition matches any set of rows whose first row meets the definition of symbol A, followed by a non-empty sequence of rows, each of which meets the definition of either symbol B or C, optionally followed by one row that meets the definition of symbol D, followed by any number of rows that meet the definition of symbol X, and ending with a row that meets the definition of symbol A.

You can use parentheses to define precedence rules. Parentheses are recommended for clarity, even where not strictly required.

## Start Anchor and End Anchor Pattern Operators

To indicate that a sequence of rows must start or end with a row that matches a certain symbol, use the start anchor (^) or end anchor (\$) operator.

Operator	Description
^A	Appears only at beginning of pattern. Indicates that set of rows must start with row that meets definition of A.
A\$	Appears only at end of pattern. Indicates that set of rows must end with row that meets definition of A.

## Subpattern Operators

Subpattern operators let you specify how often a subpattern must appear in a match. You can specify a minimum number, exact number, or range. In the following table, X represents any pattern definition composed of symbols and any of the previously described pattern operators.

Operator	Description
(X){a}	Matches exactly a occurrences of pattern X.
(X){a,}	Matches at least a occurrences of pattern X.

Operator	Description
(X){a,b}	Matches at least a and no more than b occurrences of pattern X.

## nPath Greedy Pattern Matching

The nPath function uses greedy pattern matching, finding the longest available match despite any nongreedy operators in the pattern.

For example, consider the input table link2:

link2

userid	title1	startdate	enddate
21	Chief Exec Officer	1994-10-01	2005-02-28
21	Software Engineer	1996-10-01	2001-06-30
21	Software Engineer	1998-10-01	2001-06-30
21	Chief Exec Officer	2005-03-01	2007-03-31
21	Chief Exec Officer	2007-06-01	null

This query returns the following table:

```
SELECT job_transition_path, count(*) AS count1 FROM nPath@coprocessor (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*')
  Symbols (title1 ilike 'software eng%' AS ENGR,
    true AS OTHER,
    title1 ilike 'Chief Exec Officer' AS CEO)
  Result (accumulate(title1 OF ANY(ENGR,OTHER,CEO))
    AS job_transition_path)
) AS dt GROUP BY 1;
```

job_transition_path	count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, and OTHER\* matches the remaining rows:

title	pattern('CEO.ENGR.OTHER*')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

This query returns the following table:

```
SELECT job_transition_path, count(*) AS count1 FROM nPath@coprocessor (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*.CEO')
  Symbols (title1 ilike 'software eng%' AS ENGR,
    true AS OTHER,
    title1 ilike 'Chief Exec Officer' AS CEO)
  Result (accumulate(title1 of ANY(ENGR,OTHER,CEO))
    AS job_transition_path)
) AS dt GROUP BY 1;
```

job_transition_path	count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, OTHER\* matches the next two rows, and CEO matches the last row:

title	pattern('CEO.ENGR.OTHER*.CEO')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Filters

The Filter syntax element specifies filters to impose on the matched rows. Filtering out most matches can improve performance, but memory fragmentation can degrade it. Memory fragmentation can occur in these cases:

- The mode is NONOVERLAPPING and the pattern includes the end anchor operator (\$) but not the start anchor operator (^).
- The mode is OVERLAPPING and the pattern does not include the start anchor operator.
- The first symbol in the pattern can match an infinite number of input rows.
- The data partition is huge.
- The Java Virtual Machine (JVM) is too small.

If nPath runs much slower with the Filter syntax element, increase the size of the JVM. If the problem persists, alter the pattern.

## nPath Filters Example

Using clickstream data from an online store, this example finds the sessions where the user visited the checkout page within 10 minutes of visiting the home page. Because there is no way to know in advance how many rows might appear between the home page and the checkout page, the example cannot use a LAG or LEAD expression. Therefore, it uses the Filter syntax element.

### Input

clickstream

userid	sessionid	clicktime	pagetype
1	1	10-10-2012 10:15	home
1	1	10-10-2012 10:16	view
1	1	10-10-2012 10:17	view
1	1	10-10-2012 10:20	checkout
1	1	10-10-2012 10:30	checkout
1	1	10-10-2012 10:35	view
1	1	10-10-2012 10:45	view
2	2	10-10-2012 13:15	home
2	2	10-10-2012 13:16	view
2	2	10-10-2012 13:43	checkout
2	2	10-10-2012 13:35	view
2	2	10-10-2012 13:45	view

### SQL Call

```
SELECT * FROM nPath@coprocessor (
  ON clickstream PARTITION BY userid ORDER BY clicktime
  USING
```

```

Symbols (pagetype='home' AS home,
        pagetype!='home' AND pagetype!='checkout' AS view,
        pagetype='checkout' AS checkout)
Pattern ('home.view*.checkout')
Result (FIRST(userid of ANY(home, checkout, view)) AS userid,
        FIRST (sessionid of ANY(home, checkout, view)) AS sessionid,
        COUNT (*) of any(home, checkout, view)) AS cnt,
        FIRST (clicktime of ANY(home)) AS firsthome,
        LAST (clicktime of ANY(checkout)) AS lastcheckout)
Filter (FIRST (clicktime + '10 minutes' ::interval OF ANY (home)) >
        FIRST (clicktime of any(checkout)))
Mode (NONOVERLAPPING)
) AS dt;

```

## Output

userid	sessionid	cnt	firsthome	lastcheckout
1	1	4	2012-10-10 10:15:00	2012-10-10 10:20:00

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Results

The Result syntax element defines the output columns, specifying the values to retrieve from the matched rows and the aggregate function to apply to these values.

For each pattern, the nPath function can apply one or more aggregate functions to the matched rows and output the aggregated results.

Supported aggregate functions:

- SQL aggregate functions AVG, COUNT, MAX, MIN, and SUM
- ML Engine nPath sequence aggregate functions described in the following table

In the following table, *col\_expr* is an expression whose value is a column name, *symbol* is defined by the Symbols syntax element, and *symbol\_list* has this syntax:

```
{ symbol | ANY (symbol[,...]) }
```

Function	Description
COUNT (         { *   [DISTINCT] col_expr }         OF symbol_list )	Returns either the number of total number of matched rows (*) or the number (or distinct number) of col_expr values in the matched rows.

Function	Description
FIRST ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the <i>col_expr</i> value of the first matched row.
LAST ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the <i>col_expr</i> value of the last matched row.
NTH ( <i>col_expr</i> , <i>n</i> OF <i>symbol_list</i> )	Returns the <i>col_expr</i> value of the <i>n</i> th matched row, where <i>n</i> is a nonzero value of the data type SMALLINT, INTEGER, or BIGINT. The sign of <i>n</i> determines whether the <i>n</i> th matched row is <i>n</i> th from the first or last matched row. For example, if <i>n</i> is 1, the <i>n</i> th matched row is the first matched row, and if <i>n</i> is -1, the <i>n</i> th matched row is the last matched row. If <i>n</i> is greater than the number of matched rows, the <i>n</i> th function returns NULL.
FIRST_NOTNULL ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the first non-null <i>col_expr</i> value in the matched rows.
LAST_NOTNULL ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the last non-null <i>col_expr</i> value in the matched rows.
MAX_CHOOSE ( <i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i> )	Returns the <i>descriptive_col_expr</i> value of the matched row with the highest-sorted <i>quantifying_col_expr</i> value. For example, MAX_CHOOSE (product_price, product_name OF B) returns the product_name of the most expensive product in the rows that map to B. The <i>descriptive_col_expr</i> can have any data type. The <i>quantifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
MIN_CHOOSE ( <i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i> )	Returns the <i>descriptive_col_expr</i> value of the matched row with the lowest-sorted <i>quantifying_col_expr</i> value. For example, MIN_CHOOSE (product_price, product_name OF B) returns the product_name of the least expensive product in the rows that map to B. The <i>descriptive_col_expr</i> can have any data type. The <i>quantifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
DUPCOUNT ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in the immediately preceding matched row. When <i>col_expr</i> is also the ORDER BY <i>col_expr</i> , this function returns the equivalent of ROW_NUMBER() - RANK().
DUPCOUNTCUM ( <i>col_expr</i> OF <i>symbol_list</i> )	Returns the cumulative duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in all preceding matched rows. When <i>col_expr</i> is also the ORDER BY <i>col_expr</i> , this function returns the equivalent of ROW_NUMBER() - DENSE_RANK().

Function	Description
<pre> ACCUMULATE (   [ DISTINCT     CDISTINCT ]   col_expr OF symbol_list   [ DELIMITER   'delimiter' ] ) </pre>	<p>Returns, for each matched row, the concatenated values in <i>col_expr</i>, separated by <i>delimiter</i>. The default delimiter is ',' (a comma followed by a space).</p> <p>DISTINCT limits the concatenated values to distinct values.</p> <p>CDISTINCT limits the concatenated values to consecutive distinct values.</p>

You can compute an aggregate over more than one symbol. For example, `SUM (val OF ANY (A,B))` computes the sum of the values of the attribute *val* across all rows in the matched segment that map to A or B.

## nPath Results Examples

### nPath Results Example: FIRST, LAST\_NOTNULL, MAX\_CHOOSE, MIN\_CHOOSE

#### Input

trans1

userid	gender	ts	productname	productamt
1	M	2012-01-01 00:00:00	shoes	100
1	M	2012-02-01 00:00:00	books	300
1	M	2012-03-01 00:00:00	television	500
1	M	2012-04-01 00:00:00	envelopes	10
2		2012-01-01 00:00:00	bookcases	150
2		2012-02-01 00:00:00	tables	250
2	F	2012-03-01 00:00:00	appliances	1500
3	F	2012-01-01 00:00:00	chairs	400
3	F	2012-02-01 00:00:00	cellphones	600
3	F	2012-03-01 00:00:00	dvds	50

#### SQL Call

```

SELECT * FROM nPath@coprocessor (
  ON trans1 PARTITION BY userid ORDER BY ts
  USING
  Mode (nonoverlapping)

```

```

Pattern ('A+')
Symbols (TRUE AS A)
Result (FIRST(userid OF A) AS Userid,
        LAST_NOTNULL (gender OF A) AS Gender,
        MAX_CHOOSE (productamt, productname OF A) AS Max_prod,
        MIN_CHOOSE (productamt, productname OF A) AS Min_prod)
) AS dt;

```

## Output

userid	gender	max_prod	min_prod
1	M	television	envelopes
2	F	appliances	bookcases
3	F	cellphones	dvds

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Results Example: FIRST, ACCUMULATE

### Input

#### clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	null	home	06:59:13	Company1	100
1039	1	null	home	07:00:10	Company3	300
1039	1	television	checkout	07:00:12	Company3	500
1039	1	television	checkout	07:00:18	Company3	10
1039	1	envelopes	checkout	07:01:00	Company4	10
1039	1	null	checkout	07:01:10	Company4	10

### SQL Call

```

SELECT * FROM nPath@coprocessor (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode ('nonoverlapping')
  Symbols (pagetype='home' AS H, pagetype='checkout' AS C,
           pagetype!='home' AND pagetype!='checkout' AS A)

```

```

Pattern ('^H+.A*.C+$')
Result (
  FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
  FIRST (clicktime OF H) AS firsthome,
  FIRST (clicktime OF C) AS firstcheckout,
  ACCUMULATE (productname OF ANY (H,A,C) DELIMITER '*')
    AS products_accumulate,
  ACCUMULATE (CDISTINCT productname OF ANY (H,A,C) DELIMITER '$$')
    AS cde_dup_products,
  ACCUMULATE (DISTINCT productname OF ANY (H,A,C))
    AS de_dup_products
)
) AS dt;

```

## Output

sessionid	firsthome	firstcheckout	products_accumulate	cde_dup_products	de_dup_products
1	06:59:13	07:00:12	[null*null*television*television*envelopes*null]	[null\$\$television\$\$envelopes\$\$null]	[null, television, envelopes]

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Results Example: FIRST, ACCUMULATE, COUNT, NTH

### Input

The input table is clicks, as in [nPath Results Example: FIRST, ACCUMULATE](#).

### SQL Call

```
SELECT * FROM nPath@coprocessor (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode ('nonoverlapping')
  Symbols (pagetype='home' AS H, pagetype='checkout' AS C,
           pagetype!='home' AND pagetype!='checkout' AS A)
  Pattern ('^H+.A*.C+$')
  Result (
    FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
    FIRST (clicktime OF H) AS firsthome,
    FIRST (clicktime OF C) AS firstcheckout,
    ACCUMULATE (productname OF ANY (H,A,C))
      AS products_accumulate,
    COUNT (DISTINCT productname OF ANY(H,A,C))
      AS count_distinct_products,
    ACCUMULATE (CDISTINCT productname OF ANY (H,A,C))
      AS consecutive_distinct_products,
    ACCUMULATE (DISTINCT productname OF ANY (H,A,C))
      AS distinct_products,
    NTH (productname, -1 OF ANY(H,A,C)) AS nth
  )
) AS dt;
```

## Output

sessionid	firsthome	firstcheckout	products_accumulate	count_distinct_products	consecutive_distinct_products	distinct_products	nth
1	06:59:13	07:00:12	[null, null, television, television, envelopes, null]	3	[null, television, envelopes, null]	[null, television, envelopes]	null

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Examples

### nPath Example: Pages Visited in Each Session

This example accumulates the pages visited in each session.

#### Input

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

#### SQL Call

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode(nonoverlapping)
    Pattern('A*')
    Symbols(true AS A)
    Result(first(sessionid of A) AS sessionid,
           accumulate(pagetype of A) AS path)
) AS dt;
```

#### Output

```
sessionid
path
-----
-----
4 [home, home, home, home, home, home, home1, home1, home1, page1, page1,
page1]
2 [home, home, home, home, home, home, home, home, home, home1, page1,
checkout, checkout, home, home]
3 [home, home, home, home, home, home, home, home, home1, page1, home,
home1, page1, home]
1 [home, home1, page1, home, home1, page1, home, home, home, home1, page1,
checkout, home, home, home, home, home, home, home, home]
5 [home, home, home, home, home1, home1, home1, page1, page1, page1,
page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]
```

### nPath Example: Sessions Start at Home and Visit Page1

This example finds the sessions that start at the home page and visit Page1.

## Input

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

## SQL Call

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode(nonoverlapping)
    Pattern('^H.A*.P1.A*')
    Symbols(pagetype='home' AS H, pagetype='page1' AS P1, TRUE AS A)
    Result(FIRST(sessionid OF A) AS sessionid,
           accumulate(pagetype OF ANY(H,P1,A)) AS path)
) AS dt;
```

## Output

```
sessionid
path
-----
-----
-----
4 [home, home, home, home, home, home, home1, home1, home1, page1, page1,
page1]
2 [home, home, home, home, home, home, home, home, home, home1, page1,
checkout, checkout, home, home]
5 [home, home, home, home, home1, home1, home1, page1, page1, page1,
page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]
1 [home, home1, page1, home, home1, page1, home, home, home, home1, page1,
checkout, home, home, home, home, home, home, home, home]
3 [home, home, home, home, home, home, home, home, home1, page1, home,
home1, page1, home]
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Example: Checkout Paths for Purchases Over \$200

This example finds the paths to the checkout page for purchases over \$200.

**Input**

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

**SQL Call**

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode(nonoverlapping)
    Pattern('A*.C+.A*')
    Symbols(productprice > 200 AND pagetype='checkout' AS C, true AS A)
    Result(first(sessionid of A) AS sessionid,
           accumulate(pagetype OF ANY(A,C)) AS path,
           AVG(productprice OF ANY(A,C)) AS sum)
) AS dt;
```

**Output**

```
sessionid
path
sum
-----
-----
-----
1 [home, home1, page1, home, home1, page1, home, home, home, home1, page1,
checkout, home, home, home, home, home, home, home, home, home] 602.8571428571429
5 [home, home, home, home, home1, home1, home1, page1, page1, page1,
page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]
363.1578947368421
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

**nPath Example: Mode (OVERLAPPING)****Input**

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

**SQL Call**

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
```

```

Mode (overlapping)
Pattern ('A.A')
Symbols (TRUE AS A)
Result (FIRST(sessionid OF A) AS sessionid,
        accumulate(pagetype OF A) AS path)
) AS dt ORDER BY sessionid;

```

## Output

```

sessionid path
-----
1 [home, home]
1 [home, home]
1 [home, home]
1 [home, home]
1 [checkout, home]
1 [page1, checkout]
1 [home1, page1]
1 [home, home1]
1 [home, home]
1 [home, home]
1 [page1, home]
1 [home, home1]
1 [page1, home]
1 [home1, page1]
1 [home, home1]
1 [home1, page1]
1 [home, home]
1 [home, home]
1 [home, home]
1 [home, home]
2 [checkout, checkout]
2 [home1, page1]
2 [home, home1]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [home, home]
2 [page1, checkout]
2 [checkout, home]

```

```
2 [home, home]
3 [home, home1]
3 [home1, page1]
3 [home, home1]
3 [home, home]
3 [home, home]
3 [home, home]
3 [home, home]
3 [home, home]
3 [home, home]
3 [home, home]
3 [page1, home]
3 [home1, page1]
3 [page1, home]
4 [home1, page1]
4 [home1, home1]
4 [home, home1]
4 [home, home]
4 [home, home]
4 [home, home]
4 [home, home]
4 [home, home]
4 [home1, home1]
4 [page1, page1]
4 [page1, page1]
5 [checkout, page2]
5 [checkout, checkout]
5 [page2, checkout]
5 [page2, page2]
5 [page1, page2]
5 [page1, page1]
5 [page1, page1]
5 [home1, page1]
5 [home1, home1]
5 [home1, home1]
5 [home, home1]
5 [home, home]
5 [home, home]
5 [home, home]
5 [page2, page2]
5 [checkout, checkout]
5 [page2, page2]
5 [page2, page2]
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Example: First Product with Multiple Referrers

This example finds the first product with multiple referrers in any session.

### Input

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

### SQL Call

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode(nonoverlapping)
    Pattern('REFERRER{2,}')
    Symbols(referrer IS NOT NULL AS REFERRER)
    Result (FIRST(sessionid OF REFERRER) AS sessionid,
            FIRST(productname OF REFERRER) AS product)
) AS dt;
```

### Output

```
sessionid product
-----
          5 appliances
          4 tables
          2 tables
          3 bookcases
          1 envelopes
```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Example: Sessions that Checked 3-6 Products

This example finds the data for sessions that checked three to six products.

For sessions where the user checked between three and six products (exclusive), return the names of the most and least expensive products, the maximum price of the most expensive product, and the minimum price of the least expensive product.

**Input**

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

**SQL Call**

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode(nonoverlapping)
    Pattern('H+.D*.C{3,6}.D')
    Symbols(pagetype = 'home' AS H, pagetype='checkout' AS C,
            pagetype<>'home' AND pagetype<>'checkout' AS D)
    Result(first(sessionid of C) AS sessionid,
            max_choose(productprice, productname of C) AS most_expensive_product,
            max(productprice of C) AS max_price,
            min_choose(productprice, productname of C) AS least_expensive_product,
            min(productprice of C) AS min_price)
) AS dt;
```

**Output**

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600.0	bookcases	150.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

**nPath Example: Sessions that Checked at Least 3 Products**

This example finds the data for sessions that checked at least three products.

Modify the SQL call in [nPath Example: Sessions that Checked 3-6 Products](#) to find sessions where the user checked at least three products by changing the Pattern syntax element.

**Input**

- aggregate\_clicks, as in [LAG and LEAD Expressions Example: First and Most Expensive Purchases](#) (under [nPath Symbols](#))

**SQL Call**

```
SELECT * FROM NPath@coprocessor (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
```

```

Mode(nonoverlapping)
Pattern('H+.D*.C{3,}.D')
Symbols(pagetype = 'home' AS H, pagetype='checkout' AS C,
        pagetype<>'home' AND pagetype<>'checkout' AS D)
Result(first(sessionid of C) AS sessionid,
        max_choose(productprice, productname of C) AS most_expensive_product,
        max(productprice of C) AS max_price,
        min_choose(productprice, productname of C) AS least_expensive_product,
        min(productprice of C) AS min_price)
) AS dt;

```

## Output

```

sessionid most_expensive_product max_price least_expensive_product min_price
-----
5 cellphones 600.0 bookcases 150.0

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## nPath Example: Multiple Partitioned Input Tables and Dimension Input Table

An e-commerce store wants to count the advertising impressions that lead to a user clicking an online advertisement. The example counts the online advertisements that the user viewed and the television advertisements that the user might have viewed.

## Input

### impressions

userid	ts	imp
1	2012-01-01	ad1
1	2012-01-02	ad1
1	2012-01-03	ad1
1	2012-01-04	ad1
1	2012-01-05	ad1
1	2012-01-06	ad1
1	2012-01-07	ad1
2	2012-01-08	ad2
2	2012-01-09	ad2

userid	ts	imp
2	2012-01-10	ad2
2	2012-01-11	ad2
...	...	...

**clicks2**

userid	ts	click
1	2012-01-01	ad1
2	2012-01-08	ad2
3	2012-01-16	ad3
4	2012-01-23	ad4
5	2012-02-01	ad5
6	2012-02-08	ad6
7	2012-02-14	ad7
8	2012-02-24	ad8
9	2012-03-02	ad9
10	2012-03-10	ad10
11	2012-03-18	ad11
12	2012-03-25	ad12
13	2012-03-30	ad13
14	2012-04-02	ad14
15	2012-04-06	ad15

**tv\_spots**

ts	tv_imp
2012-01-01	ad1
2012-01-02	ad2
2012-01-03	ad3
2012-01-04	ad4
2012-01-05	ad5
2012-01-06	ad6

ts	tv_imp
2012-01-07	ad7
2012-01-08	ad8
2012-01-09	ad9
2012-01-10	ad10
2012-01-11	ad11
2012-01-12	ad12
2012-01-13	ad13
2012-01-14	ad14
2012-01-15	ad15

## SQL Call

The tables impressions and clicks have a user\_id column, but the table tv\_spots is only a record of television advertisements shown, which any user might have seen. Therefore, tv\_spots must be a dimension table.

```
SELECT * FROM NPath@coprocessor (
  ON impressions PARTITION BY userid ORDER BY ts
  ON clicks2 PARTITION BY userid ORDER BY ts
  ON tv_spots DIMENSION ORDER BY ts
  USING
    Mode('nonoverlapping')
    Symbols(true AS imp, TRUE AS click, TRUE AS tv_imp)
    Pattern('(imp|tv_imp)*.click')
    Result (COUNT(* OF imp) AS imp_cnt,
            COUNT(* OF tv_imp) AS tv_imp_cnt)
) AS dt;
```

## Output

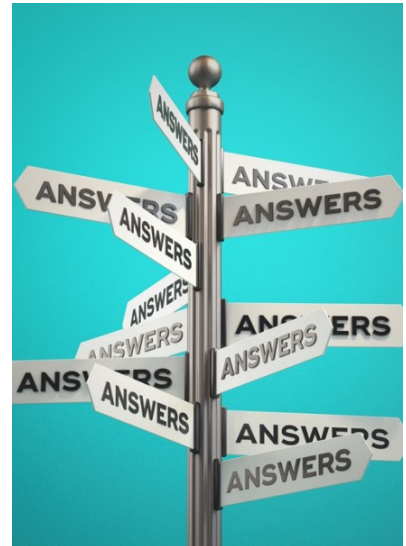
```
imp_cnt tv_imp_cnt
-----
      23          0
      19          0
      24          0
      22          0
      23          0
      22          0
      19          0
```

23	0
18	0
22	0
20	0
25	0
21	0
22	0
22	0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## Sessionize Background Information

- **Sessionize**
  - **Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)**
  - Labs
  - Review
- Attribution
  - Background Information (Description and Use Cases)
  - Single-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs)
  - Multiple-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs) - Optional
  - Review



## Sessionize Description

- The **Sessionize** function maps each click in a session to a unique session identifier
- A "session" is defined as a sequence of clicks by one user that are separated by at most **n** seconds
- The function is useful both for sessionization and for detecting web crawler (bot) activity
- It is typically used to understand user browsing behavior on a web site

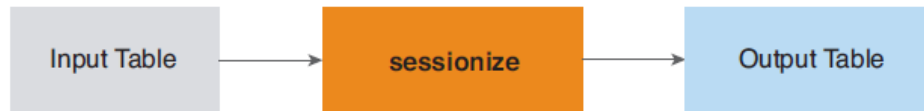
The **Sessionize** function maps each click in a session to a unique session identifier.  
A "session" is defined as a sequence of clicks by one user that are separated by at most **n** seconds.  
The function is useful both for sessionization and for detecting web crawler (bot) activity.  
It is typically used to understand user browsing behavior on a web site.

## Sessionize Use Case Examples

- A **Retailer** wishes to know which pages on its website are visited in the most sessions
- A **Banking institution** wishes to know if there have been any attempted bot infiltrations into customer accounts
- A **Social-media website** wishes to sell advertising space and wants to know the number of sessions each user has per day, and the average length in time of those sessions

Sessionize can be used whenever you wish to group time-based events together.

## Sessionize Workflow



- The **Sessionize** function reads data from an input table, view, or query, and then outputs **sessionid** (per specified arguments)
- For example, if a **userid** has 2 consecutive clicks within 1 minute of each other, consider that the same "session"
- If > 1 minute, then increment **sessionid** counter by 1

The **Sessionize** function outputs a **sessionid** column. Note that **sessionid** always begins at **0** with each new partition

### Input

timestamp	userid	...
10:00:00	10	...
00:58:24	76	
10:00:24	10	
02:30:33	76	
10:01:23	10	
10:02:40	10	

### Output

timestamp	userid	...	sessionid
10:00:00	10	...	0
10:00:24	10		0
10:03:00	10		1
10:05:30	10		2
00:59:24	76		0
02:30:33	76		1

UserId 10 has three 'sessions': 0, 1, and 2

UserId 76 has two 'sessions': 0 and 1

**Sessionize** requires at least one input table, view, or query. Rows that meet the condition of your logic are then output, together with a new column, **SESSIONID**.

## Sessionize Syntax

```

SELECT * FROM Sessionize[@coprocessor]
(ON { table | view | (query) }
PARTITION BY expression [,...]
ORDER BY order_column
USING
TimeColumn ('timestamp_column')
TimeOut ('session_timeout')
[ ClickLag ('min_human_click_lag') ]
[ EmitNull ({'true'|'t' | 'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
) as alias;

```

Note: Sessionize can be run in the Adv SQL Engine or the ML Engine

Following are important points to realize about the syntax for **Sessionize**.

1. As with other Teradata Vantage functions, we are invoking the function through the call **SELECT \* FROM function\_name**; i.e., in this case, **SELECT \* FROM Sessionize**.
2. Our input data can be in the form of a table, view, or query. It follows the **ON** keyword.
3. We must specify which columns to use for our **PARTITION BY** and **ORDER BY** arguments.
4. Following the **USING** keyword, we are afforded the opportunity of specifying our *required* and *optional* arguments specific to the function.

The *required* arguments for **Sessionize** follow:

- **TimeColumn**: Specify the name of the input column that contains the click times. Note: The **timestamp\_column** must also be an **order\_column**.
- **TimeOut**: Specify the number of seconds at which the session times out. If **session\_timeout** seconds elapse after a click, the next click starts a new session. The data type of **session\_timeout** is DOUBLE PRECISION.

The *optional* arguments for **Sessionize** follow:

- **ClickLag** [Optional]: Specify the minimum number of seconds between clicks for the session user to be considered human. If clicks are more frequent, indicating that the user is a bot, the function ignores the session. The **min\_human\_click\_lag** must be less than **session\_timeout**. The data type of **min\_human\_click\_lag** is DOUBLE PRECISION. Default behavior: The function ignores no session, regardless of click frequency.
- **EmitNull** [Optional]: Specify whether to output rows that have NULL values in their **SESSIONID** and **CLICKLAG** columns, even if their **timestamp\_column** has a NULL value. Default: 'false'.

## Sessionize Required Arguments

- **TimeColumn:** Specify the name of the input column that contains the click times. **Note:** The `timestamp_column` must also be an `order_column`
- **TimeOut:** Specify the number of seconds at which the session times out. If `session_timeout` seconds elapse after a click, the next click starts a new session. The data type of `session_timeout` is DOUBLE PRECISION

Other than your input table/view/query, SESSIONIZE has only two required arguments:

- TimeColumn
- TimeOut

## Sessionize Optional Arguments

- **ClickLag** [Optional]: Specify the minimum number of seconds between clicks for the session user to be considered human. If clicks are more frequent, indicating that the user is a bot, the function ignores the session. The `min_human_click_lag` must be less than `session_timeout`. The data type of `min_human_click_lag` is DOUBLE PRECISION. Default behavior: The function ignores no session, regardless of click frequency
- **EmitNull** [Optional]: Specify whether to output rows that have NULL values in their session id and rapid fire columns, even if their `timestamp_column` has a NULL value. Default: 'false'

The following are optional arguments for SESSIONIZE:

- ClickLag
- EmitNull

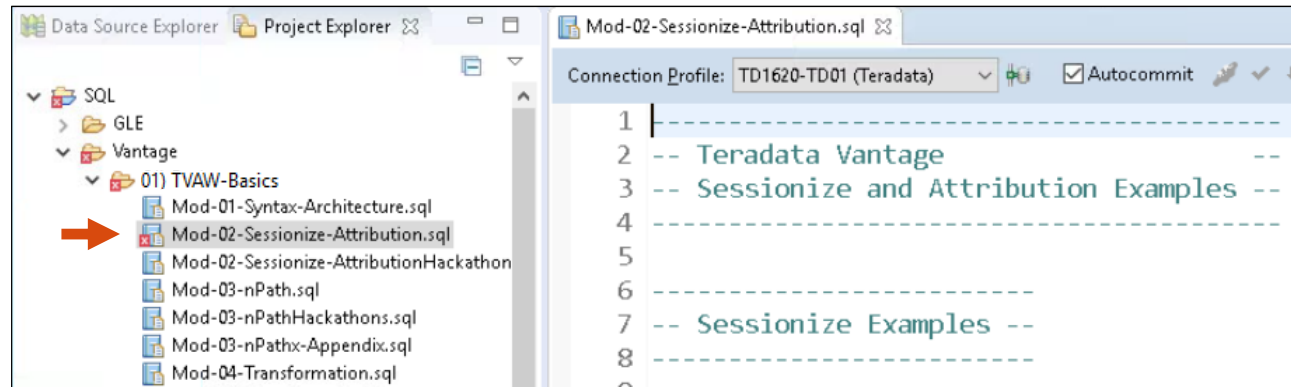
## Current Topic – Sessionize Labs

- **Sessionize**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
- **Labs**
  - Review
- Attribution
  - Background Information (Description and Use Cases)
  - Single-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs)
  - Multiple-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs) - Optional
  - Review



## Before We Begin: Open TD Studio File in Project Explorer

Navigate to 'Project Explorer' tab, then drill down to **SQL > GLE > Vantage > 01) TVAW-Basics** and double-click on: **Mod-02-Sessionize-Attribution.sql** file



We'll be running labs from Teradata Studio. Double-click on the 'Mod-02-Sessionize-Attribution.sql' file to get started.



## Lab 01a: Sessionize Intro - Input Data

**Goal:** Sessionize below data to count how many visits each userid had to a website

```
SELECT * FROM sessionme;
```

Input  
table

	userid	clicktime	productid	pagetype	referrer	price
1	578	2011-01-05 11:12:00.000000	bose	checkout	null	340.00
2	578	2011-01-05 11:13:00.000000	ipad	checkout	null	450.00
3	333	2011-01-05 09:11:00.000000	ipod	checkout	www.yahoo.com	200.20
4	333	2011-01-05 09:12:00.000000	bose	checkout	null	340.00
5	333	2011-01-05 09:10:00.000000	null	home	www.yahoo.com	null
6	333	2011-01-05 09:13:00.000000	null	home	www.google.com	null
7	333	2011-01-05 09:14:00.000000	null	home	www.google.com	null
8	333	2011-01-05 10:06:00.000000	null	home	www.google.com	null
9	333	2011-01-05 10:07:00.000000	null	home	null	null
10	333	2011-01-05 10:08:00.000000	null	home	null	null
11	333	2011-01-05 10:09:00.000000	iphone	checkout	null	650.00
12	578	2011-01-05 11:10:00.000000	bose	checkout	null	750.00
13	578	2011-01-05 11:11:00.000000	null	home	www.godaddy.com	null
14	578	2011-01-05 11:14:00.000000	iphone	checkout	null	650.00

Here, we are viewing the contents of the sessionme table.



## Lab 01b: Sessionize – Teradata Vantage using Adv SQL Engine (1 of 2)

teradata.

**Query:** Sessionize User's clicks that are within 1 minute of each other

### Sessionize query

```
SELECT * FROM Sessionize
(ON sessionme
PARTITION BY userid
ORDER BY clicktime
USING
TimeColumn ('clicktime')
TimeOut (60)
ClickLag (0.2)
EmitNull ('false')
) order by userid, clicktime;
```

### Input table

	userid	clicktime	productid	pagetype	referrer	price
1	578	2011-01-05 11:12:00.000000	bose	checkout	null	340.00
2	578	2011-01-05 11:13:00.000000	ipad	checkout	null	450.00
3	333	2011-01-05 09:11:00.000000	ipod	checkout	www.yahoo.com	200.20
4	333	2011-01-05 09:12:00.000000	bose	checkout	null	340.00
5	333	2011-01-05 09:10:00.000000	null	home	www.yahoo.com	null
6	333	2011-01-05 09:13:00.000000	null	home	www.google.com	null
7	333	2011-01-05 09:14:00.000000	null	home	www.google.com	null
8	333	2011-01-05 10:06:00.000000	null	home	www.google.com	null
9	333	2011-01-05 10:07:00.000000	null	home	null	null
10	333	2011-01-05 10:08:00.000000	null	home	null	null
11	333	2011-01-05 10:09:00.000000	iphone	checkout	null	650.00
12	578	2011-01-05 11:10:00.000000	bose	checkout	null	750.00
13	578	2011-01-05 11:11:00.000000	null	home	www.godaddy.com	null
14	578	2011-01-05 11:14:00.000000	iphone	checkout	null	650.00

Here, we are sessionizing our data. Note the values within our arguments after the USING clause.



## Lab 01b: Sessionize – Teradata Vantage using Adv SQL Engine (2 of 2) - Output

teradata.

**Query:** Sessionize User's clicks that are within 1 minute of each other

```
SELECT * FROM Sessionize
(ON sessionme
PARTITION BY userid
ORDER BY clicktime
USING
TimeColumn ('clicktime')
TimeOut (60)
ClickLag (0.2)
EmitNull ('false')
) ORDER BY userid,clicktime;
```

### Output

	userid	clicktime	productid	pagetype	referrer	price	SESSIONID	CLICKLAG
1	333	2018-01-05 09:10:00.0...	null	home	www.yaho...	null	0	f
2	333	2018-01-05 09:11:00.0...	ipod	checkout	www.yaho...	200.2	0	f
3	333	2018-01-05 09:12:00.0...	bose	checkout	null	340	0	f
4	333	2018-01-05 09:13:00.0...	null	home	www.goo...	null	0	f
5	333	2018-01-05 09:14:00.0...	null	home	www.goo...	null	0	f
6	333	2018-01-05 10:06:00.0...	null	home	www.goo...	null	1	f
7	333	2018-01-05 10:07:00.0...	null	home	null	null	1	f
8	333	2018-01-05 10:08:00.0...	null	home	null	null	1	f
9	333	2018-01-05 10:09:00.0...	iphone	checkout	null	650	1	f
10	578	2018-01-05 11:10:00.0...	bose	checkout	null	750	0	f
11	578	2018-01-05 11:11:00.0...	null	home	www.god...	null	0	f
12	578	2018-01-05 11:12:00.0...	bose	checkout	null	340	0	f
13	578	2018-01-05 11:13:00.0...	ipad	checkout	null	450	0	f
14	578	2018-01-05 11:14:00.0...	iphone	checkout	null	650	0	f

Note the following:

- **userid 333** had two visits, denoted by **sessionid** values **0** and **1**
- **userid 578** had one visit, denoted by **sessionid** value **0**

We have partitioned by **user\_id** and ordered by **clicktime**.

Any **user\_id** clicks that occur within 60 seconds of one another will be in the same **SESSIONID**.

The value for **SESSIONID** restarts at **0** for each change in **user\_id**.



## Lab 02a - Understanding the Data (1 of 2)

- Here, we familiarize ourselves with the **bank\_web\_clicks** table
- The next many slides will walk through the data and various examples of using the **Sessionize** syntax against this table

ANSI SQL

```
SELECT * FROM bank_web_clicks
WHERE customer_id IN (8263, 30324, 620)
ORDER BY customer_id ASC, datestamp ASC;
```

Here, we are viewing the contents of our input table.



## Lab 02a - Understanding the Data (2 of 2)

```
CREATE MULTISET TABLE USER1.bank_web_clicks,FALLBACK,  
  NO BEFORE JOURNAL,  
  NO AFTER JOURNAL,  
  CHECKSUM = DEFAULT,  
  DEFAULT MERGEBLOCKRATIO,  
  MAP = TD_MAP1  
(  
  customer_id INTEGER,  
  page VARCHAR(255) CHARACTER SET LATIN NOT CASESPECIFIC,  
  timestamp TIMESTAMP(6))  
PRIMARY INDEX ( customer_id ,timestamp )  
INDEX ( customer_id );
```

	customer_id	page	timestamp
1	620	ACCOUNT SUMMARY	2004-03-20 12:35:46.000000
2	620	ACCOUNT HISTORY	2004-03-20 12:38:56.000000
3	620	ACCOUNT HISTORY	2004-03-20 12:41:29.000000
4	620	PROFILE UPDATE	2004-03-20 12:42:51.000000
5	620	VIEW DEPOSIT DETAILS	2004-03-20 12:45:10.000000
6	8263	ACCOUNT SUMMARY	2004-03-21 20:38:54.000000
7	8263	ACCOUNT HISTORY	2004-03-21 20:42:39.000000
8	8263	PROFILE UPDATE	2004-03-21 20:44:59.000000
9	8263	FUNDS TRANSFER	2004-03-21 20:46:04.000000
10	30324	ACCOUNT SUMMARY	2004-05-01 15:03:13.000000
11	30324	ONLINE STATEMENT ENROLLMENT	2004-05-01 15:07:06.000000
12	30324	ACCOUNT SUMMARY	2004-05-01 15:10:43.000000
13	30324	FAQ	2004-05-01 15:12:12.000000
14	30324	ACCOUNT HISTORY	2004-05-01 15:15:15.000000

- For each customer, we know which webpage they visited and when they visited it
- Note that the **timestamp** column is of data type **TIMESTAMP**

Note that we have customer IDs, which page they visited, and when they visited it.



## Lab 02b - Required Arguments and Output (1 of 2)

teradata.

```
SELECT * FROM Sessionize  
(ON bank_web_clicks  
PARTITION BY customer_id  
ORDER BY datestamp  
USING  
TimeColumn ('datestamp')  
TimeOut (600)  
) order by customer_id,  
datestamp;
```

- The **ON** clause contains the input table
- The **PARTITION BY** argument specifies for each distinct instance of **customer\_id**, the sessionize function will re-start at a value of **0**
- The **ORDER BY** argument specifies that for each customer, data will be sessionized according to the **datestamp** value (ascending by default)
- The **USING** clause defines the **TimeColumn** (the input column that contains our timestamp data) and the user-defined **TimeOut** value (must be defined in seconds). As long as a user's clicks occur within the same **600 second window**, they will be considered as part of the same "session"

- The **ON** clause contains the input table.
- The **PARTITION BY** argument specifies that for each distinct instance of **customer\_id**, the sessionize function will re-start at a value of **0**.
- The **ORDER BY** argument specifies that for each customer, data will be sessionized according to the **datestamp** value (ascending by default).
- The **USING** clause defines the **TimeColumn** (the input column that contains our timestamp data) and the user-defined **TimeOut** value (must be defined in seconds). As long as a user's clicks occur within the same **600 second window**, they will be considered as part of the same "session".



## Lab 02b - Required Arguments and Output (2 of 2)

teradata.

	customer_id	page	timestamp	SESSIONID
1	32	ACCOUNT SUMMARY	2004-04-14 20:57:15.000000	0
2	32	VIEW DEPOSIT DETAILS	2004-04-14 21:01:07.000000	0
3	32	ACCOUNT SUMMARY	2004-04-15 10:24:53.000000	1
4	32	FUNDS TRANSFER	2004-04-15 10:28:43.000000	1
5	32	BILL MANAGER FORM	2004-04-15 10:29:20.000000	1
6	32	CUSTOMER SUPPORT	2004-04-15 10:29:54.000000	1
7	32	ACCOUNT SUMMARY	2004-04-15 15:19:29.000000	2
8	32	ACCOUNT HISTORY	2004-04-15 15:20:20.000000	2
9	32	FUNDS TRANSFER	2004-04-15 15:24:19.000000	2
10	32	ACCOUNT SUMMARY	2004-04-15 21:50:04.000000	3
11	32	VIEW DEPOSIT DETAILS	2004-04-15 21:53:19.000000	3
12	32	CUSTOMER SUPPORT	2004-04-15 21:54:10.000000	3
13	32	FUNDS TRANSFER	2004-04-15 21:54:58.000000	3
14	32	ACCOUNT SUMMARY	2004-04-16 14:18:14.000000	4
15	32	BILL MANAGER FORM	2004-04-16 14:20:34.000000	4
16	32	BILL MANAGER ENROLLMENT	2004-04-16 14:22:52.000000	4
17	32	FAQ	2004-04-16 14:25:34.000000	4
18	32	ACCOUNT SUMMARY	2004-04-16 14:28:09.000000	4
19	32	ACCOUNT SUMMARY	2004-04-16 17:49:32.000000	5
20	32	FUNDS TRANSFER	2004-04-16 17:51:04.000000	5
21	32	FUNDS TRANSFER	2004-04-16 17:54:33.000000	5
22	32	FUNDS TRANSFER	2004-04-16 17:55:10.000000	5
23	32	ACCOUNT HISTORY	2004-04-16 17:58:56.000000	5

- Here, we are viewing the output of our query from the previous page
- Note the creation of the **SESSIONID** column
- As long as the clicks of a single **customer\_id** occurred **within 600 seconds** of one another, they will share the same **SESSIONID** value

Here, we are viewing our sessionized output. Note the creation of the SESSIONID column.



## Lab 03a - Specifying a Query in the ON Clause (1 of 2)

teradata.

```
SELECT * FROM Sessionize  
(ON (SELECT * FROM bank_web_clicks  
      WHERE customer_id  
      IN (8263, 30324, 620))  
PARTITION BY customer_id  
ORDER BY timestamp  
USING  
TimeColumn ('timestamp')  
Timeout (120)  
) ORDER BY customer_id, timestamp;
```

- Note that you can also specify a query in the **ON** clause to select desired input data, as opposed to just putting the name of a table or view that contains the input data (as we did in the previous lab)
- When specifying a query, you must enclose it within parentheses
- If desired, you could write your query to **SELECT** only certain columns

The **ON** clause can be written to run the function against a sub-set of the input data source.



## Lab 03a - Specifying a Query in the ON Clause (2 of 2)

teradata.

For each **customer\_id**, as long as clicks occur within 120 seconds of one another, they will be part of the same **SESSIONID**

	customer_id	page	timestamp	SESSIONID
1	620	ACCOUNT SUMMARY	2004-03-20 12:35:46.000000	0
2	620	ACCOUNT HISTORY	2004-03-20 12:38:56.000000	1
3	620	ACCOUNT HISTORY	2004-03-20 12:41:29.000000	2
4	620	PROFILE UPDATE	2004-03-20 12:42:51.000000	2
5	620	VIEW DEPOSIT DETAILS	2004-03-20 12:45:10.000000	3
6	8263	ACCOUNT SUMMARY	2004-03-21 20:38:54.000000	0
7	8263	ACCOUNT HISTORY	2004-03-21 20:42:39.000000	1
8	8263	PROFILE UPDATE	2004-03-21 20:44:59.000000	2
9	8263	FUNDS TRANSFER	2004-03-21 20:46:04.000000	2
10	30324	ACCOUNT SUMMARY	2004-05-01 15:03:13.000000	0
11	30324	ONLINE STATEMENT ENROLLMENT	2004-05-01 15:07:06.000000	1
12	30324	ACCOUNT SUMMARY	2004-05-01 15:10:43.000000	2
13	30324	FAQ	2004-05-01 15:12:12.000000	2
14	30324	ACCOUNT HISTORY	2004-05-01 15:15:15.000000	3

Here, we are viewing the output of our SESSIONIZE query,



## Lab 04a - Detecting Bots (1 of 2)

```
SELECT * FROM Sessionize
(ON (SELECT * FROM bank_web_clicks
    WHERE customer_id IN (7172))
PARTITION BY customer_id
ORDER BY timestamp
USING
TimeColumn ('timestamp')
Timeout (60)
ClickLag (0.1)
) ORDER BY customer_id, timestamp;
```

- We can use the optional argument **ClickLag** to detect possible bot activity
- Just like **Timeout**, **ClickLag** is expressed in seconds
- Any clicks that occur within **0.1** seconds of one another will be flagged accordingly in the output



Query: Customer 7172 can't login to their on-line bank account.

Write query that will SESSIONIZE the bank\_web\_clicks table for customer\_id = 7172 with a TIMEOUT = 60 seconds and robot ClickLag = 0.10

Does anything look fishy?

Possible “bot” activity can be detected by using the optional **ClickLag** argument.



## Lab 04a - Detecting Bots (2 of 2)

customer_id	page	timestamp	SESSIONID	CLICKLAG
7172	ACCOUNT SUMMARY	2004-03-22 04:46:12.000000	0	f
7172	FUNDS TRANSFER	2004-03-22 04:48:40.000000	1	f
7172	FAQ	2004-03-22 04:50:11.000000	2	f
7172	FUNDS TRANSFER	2004-03-22 04:53:43.000000	3	f
7172	VIEW DEPOSIT DETAILS	2004-03-22 04:57:39.000000	4	f
7172	PROFILE UPDATE	2004-03-22 05:01:33.000000	5	f
...	...	...	...	...
7172	FUNDS TRANSFER	2004-03-23 20:33:34.000000	45	f
7172	VIEW DEPOSIT DETAILS	2004-03-23 20:34:46.000000	46	f
7172	VIEW DEPOSIT DETAILS	2004-03-23 20:36:59.000000	47	f
7172	FAQ	2004-03-23 20:38:07.000000	48	f
7172	LOGIN	2014-03-25 04:00:00.000000	49	f
7172	LOGIN	2014-03-25 04:00:00.100000	49	t
7172	LOGIN	2014-03-25 04:00:00.200000	49	t
7172	LOCKOUT	2014-03-25 04:00:00.300000	49	t

- The **CLICKLAG** column receives a value of 't' if a click occurred within **0.1 seconds** of the previous click
- For **SESSIONID 49**, it appears that a bot was attempting to log into the customer's bank account before being locked out

Our **ClickLag** argument has returned a positive indicator in this example.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (1 of 7)

teradata.

```
CREATE SET TABLE chips_sessionized AS
(SELECT * FROM Sessionize
(ON (SELECT remote_host, request_time,
        requested_page
        FROM chips_clean)
PARTITION BY remote_host
ORDER BY request_time asc
USING
TimeColumn ('request_time')
Timeout (3600)
)
)
WITH DATA;
```

- It will often be beneficial to land your **Sessionize** results into a physical table for further analysis and/or to serve as an input into other Teradata VANTAGE functions, such as **nPath**
- Here, we are "sessionizing" a subset of columns from the **chips\_clean** table
- Each "session" is defined as clicks made within the same window of **3,600 seconds** (one hour)

The next many pages will run through a multi-step example of sessionizing data to understand general customer behavior on a fictitious website.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (2 of 7)

teradata.

### Source Data

chips_clean
Columns
remote_host [VARCHAR(1000) Nullable]
remote_log_name [VARCHAR(1000) Nullable]
remote_user [VARCHAR(1000) Nullable]
request_time [TIMESTAMP Nullable]
requested_page [VARCHAR(1000) Nullable]
final_status [VARCHAR(1000) Nullable]
bytes_sent_CLF [VARCHAR(1000) Nullable]
referrer [VARCHAR(1000) Nullable]
request:User-agent [VARCHAR(1000) Nullable]

### Sessionized Data

chips_sessionized
Columns
remote_host [VARCHAR(1000) Nullable]
request_time [TIMESTAMP Nullable]
requested_page [VARCHAR(1000) Nullable]
SESSIONID [INTEGER Nullable]

```
SELECT * FROM chips_sessionized;
```



	remote host	request time	requested page	SESSIONID
1	66.249.67.87	2015-02-05 11:58:59.000000	/product.php?pid=17	75
2	199.241.148.12	2015-02-02 18:34:37.000000	/payment.php	0
3	173.209.212.222	2015-02-27 02:32:49.000000	/products.php?cid=1	1
4	69.42.11.132	2015-02-14 15:01:24.000000	/product.php?pid=14	0
5	66.249.64.128	2015-02-23 19:09:18.000000	/events.php?year=609&...	29

Here, we have sessionized our input data.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (3 of 7)

teradata.

Here, we are using our sessionization results to discover which are the most popular pages on our website; i.e., those visited in the greatest number of sessions

### ANSI SQL

```
SELECT requested_page,  
COUNT (DISTINCT remote_host || ' _ '  
|| sessionid) AS distinct_sessions  
FROM chips_sessionized  
GROUP BY requested_page  
HAVING distinct_sessions >= 700  
ORDER BY distinct_sessions DESC;
```

	requested_page	distinct_sessions
1	/products.php?cid=1	2397
2	/products.php?cid=6	1853
3	/contact.php	1766
4	/about.php	1468
5	/product.php?pid=34	1104
6	/products.php	975
7	/cart.php	952
8	/glutenfree.php	819
9	/locator.php	743
10	/account.php	721

Here, we are attempting to discover which our most frequently-visited pages are per the number of sessions that they were a part of.

**Note:** The contents of the **chips\_sessionized** table may be slightly different in you lab environment compared to what is shown in the slides.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (4 of 7)

teradata.

- Here, we have created a table comprised of one row per **remote\_host**, **SESSIONID**
- We have populated columns to specify general metrics about each session
- We will use this data to answer questions such as the following:
  - How many pages visited per session?
  - How many distinct pages visited per session?
  - How long in duration is each session?
  - What % of sessions contain an actual order?

**You will be running a series of CREATE TABLE statements within Teradata Studio**

### Session Data

remote_host	SESSIONID	checkouts	payments	pages	distinct_pages	min_request_time	max_request_time	session_duration
101.222.160.166	0	0	0	2	2	2014-12-31 06:56:51	2014-12-31 06:57:01	0 00:00:10.000000
162.44.245.105	0	3	1	7	5	2015-02-18 19:46:39	2015-02-18 19:55:31	0 00:08:52.000000
157.55.39.62	59	0	0	21	15	2015-02-11 13:37:01	2015-02-11 17:16:27	0 03:39:26.000000
157.55.39.178	9	0	0	1	1	2015-01-03 05:02:09	2015-01-03 05:02:09	0 00:00:00.000000
66.249.69.102	19	0	0	1	1	2015-01-09 17:33:06	2015-01-09 17:33:06	0 00:00:00.000000

Here, we have employed logic to determine various metrics about customer navigation of our website:

- How many sessions?
- How many distinct pages visited per session?
- How long in duration is each session?
- What % of sessions contain an actual order?
- Etc.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (5 of 7)

teradata.

Here, we have summarized all session data to display average metrics in aggregate

### Output

	remote hosts	sessions	avg sessions per host	avg pages	avg distinct pages	avg session duration
1	7969	20278	2.54	3.63	3.19	0 00:14:24.948861

Note: Your Output may differ than results here

Here, we are viewing aggregated results.



## Lab 05 - Landing Sessionize Results and Summarizing Findings (6 of 7)

teradata.

- Here, we have written a query to identify the number of sessions that included purchases or not
- Note that only a tiny fraction of sessions included a payment
- Note that there is a fundamental problem with "abandoned carts"

	sessions_with_payment	sessions_with_checkout	number_of_sessions
1	n	n	32615
2	n	y	181
3	y	n	14
4	y	y	252

Here, we have discovered that precious few sessions include an actual purchase. Furthermore, there seems to be a problem with regard to "abandoned carts".



## Lab 05 - Landing Sessionize Results and Summarizing Findings (7 of 7)

teradata.

- Here, we have written a query to display general metrics parsed out by whether the session included checkout and/or payment (or not)
- Note the following:
  - Few customers make purchases
  - Abandoned carts are a problem
  - People who make purchases tend to be more engaged with the website (visit more pages and revisit pages already visited)
  - Precious few customers who actually make a purchase do so more than once

sessions_with_ payment	sessions_with_ _checkout	remote _hosts	sessions	avg_sessions_ per_host	avg_pages	avg_distinct _pages	avg_session_duration
n	n	7739	19953	2.58	3.53	3.14	0 00:14:28.832907
n	y	222	233	1.05	8.68	5.74	0 00:09:49.793991
y	y	88	92	1.05	11.71	7.46	0 00:11:59.434783

We have concluded the following:

- Few customers make purchases.
- Abandoned carts are a problem.
- People who make purchases tend to be more engaged with the website (visit more pages and revisit pages already visited).
- Precious few customers who actually make a purchase do so more than once.

## Current Topic – Sessionize Review

- **Sessionize**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
- **Review**
- Attribution
  - Background Information (Description and Use Cases)
  - Single-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs)
  - Multiple-Input Models (Workflow, Syntax, Required Arguments, Optional Arguments, Input Table, Schema, Output Table Schema, Labs) - Optional
  - Review





## Hackathon: Chips Weblog Sessionize (Optional)

The following exercise is intended to provide you with further practice on using the **Sessionize** function. There is no "right" or "wrong" answer. The intent is for you to become comfortable writing queries that use **Sessionize**

1. Run a **Sessionize** query on the **chips\_clean** table, which shows user activity on a retail website. Things to think about follow:
  - What is the *nature* of the underlying data? Data types? Number of rows? What is it showing? Etc.
  - How should the data be partitioned?
  - What is a reasonable amount of time between clicks for activity to be considered as being in the same session?
  - Was there any potential/likely "bot" activity?

In this "free-form" exercise, there are no "right" or "wrong" answers. The intent is to get you to write your own **SESSIONIZE** query(ies) so as to become more comfortable with the syntax.



## Hackathon: Chips Weblog Sessionize (Possible Answer)

teradata.

```
-- Eyeball and sessionize to volatile table
```

```
show table chips_clean;
```

```
select * from chips_clean  
sample randomized allocation 200;
```

```
create volatile table x_sessionize as  
(SELECT * FROM Sessionize (  
ON (select remote_host, request_time,  
requested_page from chips_clean)  
PARTITION BY remote_host  
ORDER BY request_time asc  
USING  
TimeColumn ('request_time')  
Timeout (3600)  
EmitNull('false')  
ClickLag (0.2) ) ) with data  
on commit preserve rows;
```

```
-- How many distinct sessions?
```

```
select count (distinct remote_host || '_' ||  
SESSIONID) as sessions from x_sessionize;
```

```
-- Possible bots?
```

```
select * from x_sessionize where clicklag = 't';
```

Sessionized Data (subset)

	remote host	request time	requested p...	SESSIONID	CLICKLAG
1	54.205.142.104	2015-02-02 0...	/products.ph...	0	f
2	66.249.69.183	2015-01-01 1...	/product.php...	3	f
3	157.55.39.181	2015-02-07 0...	/product.php...	112	f
4	98.30.136.182	2015-01-25 0...	/locator.php?...0	0	f
5	66.249.65.191	2015-01-30 0...	/events.php?...14	14	f
6	66.249.65.189	2015-01-29 0...	/about	4	f
7	173.15.70.129	2015-02-17 1...	/products.ph...	0	f
8	157.55.39.46	2015-01-08 0...	/account.php...	10	f
9	188.165.15.132	2014-12-31 1...	/aboutUs.html	4	f
10	23.126.57.66	2015-01-29 0...	/original-pot...	0	f

In this “free-form” exercise, there are no “right” or “wrong” answers. The intent is to get you to write your own **SESSIONIZE** query(ies) so as to become more comfortable with the syntax.

## Sessionize Summary

In this module, you learned how to:

- Describe what the **Sessionize** function does
- Describe typical use cases for **Sessionize**
- Write **Sessionize** queries
- Interpret the output of **Sessionize** queries

The slide features a dark blue background with a teal-colored curved shape on the left side. The Teradata logo is positioned within the teal shape, and the title '4D Analytics' is in orange text to the right. Below the title is a subtitle in white text.

teradata.

## 4D Analytics

- Architecture & Practice Enablement

## Characteristics of Time Series Data

- The data that arrives is almost always recorded as a new entry
- Possibility of data being duplicated as there may not be any changes
- Data is ephemeral (Discontinuous)
- The data typically arrives in time order
- Should be able to handle high reads & writes
- Time is a primary axis (time-intervals can be either regular or irregular)
- Deletes & Updates are Rare

No Edits, No deletes, always inserts

## Time Aware - Functions

- Available in 16.20
- **\$TD\_GROUP\_BY\_TIME**
  - Use on ANY time component data
  - Works with some existing aggregation functions
    - Average, Count, Describe, Kurtosis, Maximum, Minimum, Percentile, Rank, Skew, Sum, Std. population deviation, Std. sample deviation, Population variance, Sample variance
  - Works with all new aggregation functions
    - Bottom, Top, First, Last, Delta\_T, Median, Mode, Mean absolute deviation
- **\$TD\_TIMECODE\_RANGE**
  - Defines time range
- **FILL ()**
  - Imputes missing values
    - NULLS, <Constant>, PREV/PREVIOUS, NEXT

**\$TD\_TIMECODE\_RANGE** - Its data type is period(timestamp(6) with time zone).

## Time Aware Aggregation Functions – GROUP BY TIME

### Existing Aggregate Functions

Average	Count
Describe	Kurtosis
Maximum	Minimum
Percentile	Rank
Skew	Sum
Std. population deviation	Std. sample deviation
Population variance	Sample variance

If not in the list above, then function is not time aware and cannot be used with the GROUP BY TIME clause

### New Aggregate Functions

Bottom	Delta_T
First	Last
Median	Mode
Top	Mean absolute deviation

These new aggregate functions are only invocable with the GROUP BY TIME clause

Note: there is an existing MEDIAN function out there ... but it is an Ordered Analytic / Windowed Aggregate function ... not an aggregate function. The ordered analytic function can only be invoked using the ordered analytic syntax. (See Function and Operators SQL manual)

## GROUP BY TIME Rules & Restrictions

- GROUP BY TIME and GROUP BY cannot be used together in the same query (this restriction includes GROUP BY ROLLUP, GROUP BY CUBE, and so on).
- If GROUP BY TIME is used on a non-PTI table, the USING TIMECODE clause must be included; otherwise, an error is reported.
- A supported Time Series function must be used in conjunction with a GROUP BY TIME clause; otherwise, an error is reported.
- The timebucket (which serve as the first level of grouping, if specified) are computed based on time zero. For more information about how time zero is calculated.
- Grouping is determined first by timebucket, and then by all other fields specified in the GROUP BY TIME clause (if any). A timebucket duration is required with the GROUP BY TIME clause. Failure to include it results in an error. Each GROUP BY TIME operation must have a time interval specified in the GROUP BY TIME clause. For example, in the clause GROUP BY TIME(MINUTES(15)) the time interval is 15 minutes.

OR we can have UNBOUNDED by giving GROUP BY TIME(\*).

- The HAVING clause is supported for filtering results of aggregates with the GROUP BY TIME clause.
- The QUALIFY and WITH...BY clauses are NOT supported when the GROUP BY TIME clause is present.
- The USING TIMECODE and FILL clauses are optional and may only be used with a GROUP BY TIME clause.

5

teradata.

## Exercise: Run a GROUP BY TIME analytic (1\_1)

- Want to understand average daily temperature (DRYBULB\_TEMP\_F) for the dates of 2017-01-01 and 2017-01-31
- The time column in the table is called

Example:

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME,  
AVG(TEMPERATURE)  
FROM BUOYS  
WHERE TIMECODE BETWEEN TIMESTAMP '2017-08-11 01:00:00'  
AND TIMESTAMP '2017-08-11 03:00:00'  
GROUP BY TIME( MINUTES(30))  
USING TIMECODE(TD_TIMECODE)  
ORDER BY $TD_GROUP_BY_TIME;
```

SELECT

<Time Interval Clauses>

, AVG(<Which Column>)

FROM TIMESERIES.WEATHER

WHERE

DAY\_TIME BETWEEN TIMESTAMP '2017-01-01 00:00:00'

AND TIMESTAMP '2017-02-01 00:00:00'

GROUP BY TIME (<group definition>)

USING TIMECODE(<on which timestamp>)

ORDER BY 1;

# Time Aware Analytics

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME, SENSORID, AVG(TEMPERATURE) FROM BUOYS
WHERE TIMECODE BETWEEN TIMESTAMP '2017-08-11 01:00:00' AND TIMESTAMP '2017-08-11 03:00:00'
GROUP BY TIME( MINUTES(30) AND SENSORID) USING TIMECODE(TD_TIMECODE)
ORDER BY SENSORID, $TD_GROUP_BY_TIME;
```

Timecode-Range	Group by 30 minutes	Sensor ID	Temperature
'2017-08-11 01:00:00', '2017-08-11 01:30:00'	1	22	63.5
'2017-08-11 01:30:00', '2017-08-11 02:00:00'	2	22	64.6
'2017-08-11 02:00:00', '2017-08-11 02:30:00'	3	22	65.0
'2017-08-11 02:30:00', '2017-08-11 03:00:00'	4	22	65.1
'2017-08-11 01:00:00', '2017-08-11 01:30:00'	1	23	66.4
'2017-08-11 01:30:00', '2017-08-11 02:00:00'	2	23	65.1
'2017-08-11 02:00:00', '2017-08-11 02:30:00'	3	23	64.9
'2017-08-11 02:30:00', '2017-08-11 03:00:00'	4	23	65.1

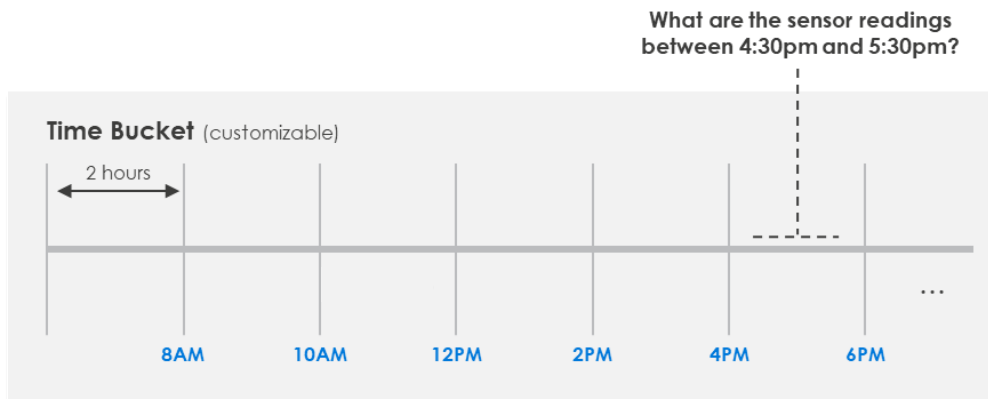
7

## Primary Time Index

- Supports time sensitive decisions
- Either Primary Index or Primary Time Index
- Fast access through:
  - Hash distribute by time bucket
  - AMP-local processing
  - Sequenced data

## Primary Time Index (PTI)

High Performance Parallelism with Efficient Storage and Access



9

teradata.

High performance parallel distribution

Customizable duration of Time Bucket provides control of distribution

Fast Primary AMP access

Customizable distribution provides AMP-local processing minimizes data movement and speeds query processing

Fully automated; set once

Data is stored in Time Order (not Hash time order)

## Primary Time Index Tables Definition

Primary Time Index (timecode\_dt, timezero\_date, bucket duration, columns, Sequenced\_Flag)

- **<timecode\_dt>**

- { TIMESTAMP | TIMESTAMP WITH TIME ZONE | DATE }
- What is the level of precision on the TIMESTAMP

Please see orange book for more details.

timecode\_dt works in combination with timebucket\_duration and timezero\_date.

- **<timezero\_date>**

- A DATE value specifying the “time zero” associated with table.
- Default timezero\_date is January 1st, 1970 @ 00:00:00 hours.

- **<timebucket\_duration>**

- A time duration specified by CAL\_YEARS, CAL\_MONTHS, CAL\_DAYS, WEEKS, DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS, MICROSECONDS.

- **<columns\_clause,>**

- COLUMNS ( <column\_list> )

- **<sequenced\_flag>**

- 10 • { SEQUENCED <optional\_maximum> | NONSEQUENCED }

teradata.

## Primary Time Index Tables (PTI)

Storage distribution choice	Time interval only	Time + column list	Column list only
	PRIMARY TIME INDEX (TIMESTAMP(0), DATE '2016-02-22', HOURS(2))	PRIMARY TIME INDEX (TIMESTAMP(2), DATE '1996-04-19', HOURS(2), COLUMNS(COUNTRYID,CARID))	PRIMARY TIME INDEX (TIMESTAMP(6), DATE '2013-01-01', COLUMNS(SENSORID))

- NON-SEQUENCED PTI table
  - Rows are stored in **time-ascending order** based on the value of the TD\_TIMECODE field.
  - Table will be **Non-sequenced by default**, if SEQUENCED/NONSEQUENCED is not specified.
- SEQUENCED PTI table
  - Rows are stored in ascending order, first based on the value of the TD\_TIMECODE field and then the TD\_SEQNO field.
  - An optional maximum value can be specified for the TD\_SEQ number value. The default maximum value is 20000.
  - Valid range of values for TD\_SEQNO is **1 to 2147483647**.

11

teradata.

These are the time series table designer options. Most of it will be easy since they should already understand the data layout and the kinds of queries most commonly needed. Most companies will start with **Time + column list** and **Time code only**. These choices have a dramatic effect on query performance. By distributing data across nodes and AMPs, we ensure parallelism. Furthermore, keys are hashed which provides the top level index without the cost of maintaining B-trees. Then, within the data blocks, there is in-table ordering which sorts the data into the timestamp order. Teradata handles sorting on column lists already so there is no need to organize the data this way when stored on disk.

## Time Series – Fill Clause

FILL SCHEME	Aggregate result for the missing time bucket will be
NULLS	Null
<Constant>	A constant value.
PREV/PREVIOUS	Same as the previous time bucket's result
NEXT	Same as the next time bucket's result

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME,  
BUOYID, AVG(TEMPERATURE) FROM OCEAN_BUOYS  
WHERE TD_TIMECODE BETWEEN  
TIMESTAMP '2017-05-02 09:45:00' AND TIMESTAMP '2017-05-02 11:45:00'  
AND SENSOR_ID=44  
GROUP BY TIME (MINUTES(15) AND BUOYID) FILL(PREV)  
ORDER BY 2,3;
```

## Teradata Time Functions

When a GROUP BY TIME query is executed,

### **TD\_GETTIMEBUCKET**

System function retrieves the TD\_TIMEBUCKET column value from a PTI table. Because a timebucket is a hash key used to determine how well the rows of a PTI table are being distributed across AMPs, avoiding skew.

### **TD\_TIME\_BUCKET\_NUMBER**

The TD\_TIME\_BUCKET\_NUMBER system function calculates the time bucket number. You can use this function with the HASHROW, HASHBUCKET, or HASHAMP functions to see how time series rows are distributed across the system.

### **TD\_TIMESERIES\_RANGE**

The TD\_TIMESERIES\_RANGE macro finds the valid ranges of the TD\_TIMECODE and TD\_SEQNO columns in a PTI table.

# Time Series

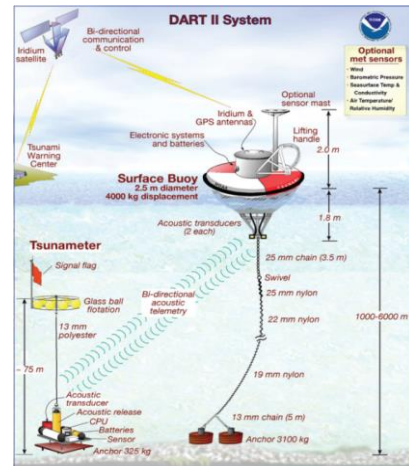
This section will cover time series.

## Time Series

**Time Series data** is data that is continuously produced and collected over a period of time

Time Series features in Vantage allows the user to capture and store Time Series data:

- Time Series data can be stored in tables with a **Primary Time Index** defined on them
- A PTI table is 'time-series' aware and provides different ways to **store and order** the time series data
- Optimized for **time range** queries
- The feature also supports **time-aware aggregate operations** using the GROUP BY TIME clause
- Useful for forecasting, detecting patterns and trends, risk reduction, etc.



Time Series data is data that is continuously produced and collected over a period of time. This kind of data is typically generated by machines such as sensors and other applications and devices that make up the Internet Of Things (IOT). Each data point in the Time Series data set is associated with a timestamp and an observed value at that time. Time Series data can be stored and analyzed to provide capabilities such as forecasting, detecting patterns and trends, anomaly detection, risk reduction etc.

The Teradata Time Series feature introduced in Teradata release 16.20 allows the user to capture and store Time Series data and perform useful aggregate operations and analytics on the data. Time Series data can be stored in tables with a new construct Primary Time Index defined on them. A Primary Time Index table is 'time-series' aware and provides different ways to store and order the time series data. The feature also supports time-aware aggregate operations that can be performed on the data set. This is done using the GROUP BY TIME clause and a set of 'time-aware' aggregate functions. All this can be combined with the existing Teradata database capabilities such as a full range of SQL support, rich collection of native and complex data types including JSON, AVRO, CSV, XML and wide range of load and extract utilities resulting in a powerful, feature-rich Time Series database offering.

## Time Series Categories

### Class I: The 7/24 infinite time series

- Hydrology: USGS uses river monitoring devices to collect time series data on all major rivers and streams
- Oceanography: There is a world-wide buoy system collecting data on a 7/24 basis
- Building Monitoring systems, Manufacturing Line Monitoring Systems

### Class II: Time Series with a “logical overlay”

- Automobile “Trip” – Start-Engine; drive from location A to location B; shut-down engine
- Plane “Flight” – Start-Engine; Take-off; Fly from location A to location B; Land; Shut-down engine
- Cargo Ship “Voyage” – Start Engine; Raise Anchor; Navigate from A to B; Drop Anchor; Shut down engine

### Class III: Fixed-size (few thousand entries) Scientific Trace Time Series

- Oil Exploration: Seismic Traces used to determine geographical sub-layers
- Medicine: Traces associated with an Ultrasound Scan or CAT Scan
- Scientific: Traces associated with the Electron Microscope to investigate crystal or cell structures

There are three categories of Time-series data.

#### Class I

Class I is 7/24 infinite time-series. In this class, data continuously collected, 7 days a week, 24 hour a day, 365 days a year...nonstop. For example, the United States Geological Survey (USGS) has Buoys in major rivers and streams collecting on a 7/24 basis and for each Buoy station, you would have a time series you keep adding data to the end. This would keep going on and on collecting until infinity.

#### Class II

With Class II data, it's a logical overlay... think about automobile... For a given trip, you start the engine and drive from point A to a destination B and then turn off the car. Overlay this trip data on the data that is collected for a given car for say #113 or air plane # 933... The importance of these trip overlays is for analysis where you may want to compare one trip to another trip for this same car #113 or compare one air plane type to another plane that does the same trip say Chicago to San Diego.

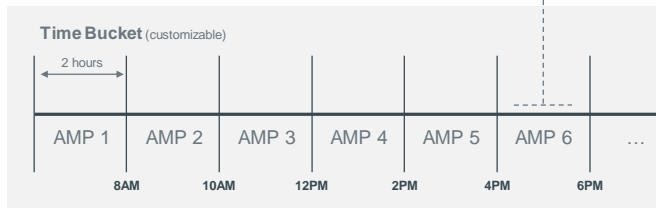
#### Class III

Class III is fixed size...Seismic Traces used to determine geographical sub-layers where the receiving electronics are setup for this discrete series. For example, data may be collected every millisecond for 6 seconds or 6,000 entries. That is the complete series... then this fixed data represents a fixed size of time series data to analyze. Other fields may have time-series data just long enough to collect an ultra-scan, Cat scan or Microscope reading to be analyzed later. No additional data is added to the end.

## Primary Time Index (PTI)

### High Performance Parallelism with Efficient Storage and Access

- High performance parallel distribution
- Customizable duration of Time Bucket provides control of distribution
- Fast Primary AMP access
- Customizable distribution provides AMP-local processing minimizes data movement and speeds query processing
- Fully automated; set once



### Three Storage Distribution Choices

Time Interval  
only

Time Interval  
and Column List

Column List only

A PTI table can be defined with 3 different data distribution strategies and 2 different ordering methods. We will discuss the distribution strategies in the next slide.

For ordering strategy, a PTI table can be defined as SEQUENCED or NONSEQUENCED in the PRIMARY TIME INDEX clause.

A PTI table is NONSEQUENCED by default i.e. if the NONSEQUENCED keyword is omitted from the PRIMARY TIME INDEX clause, the table will be considered to be non-sequenced.

A sequenced PTI table will have the auto-generated TD\_SEQNO column. The rows in the table are first ordered by TD\_TIMECODE and then by the TD\_SEQNO value. When more than one row has the same TD\_TIMECODE value, those rows are ordered by the TD\_SEQNO value within the same TD\_TIMECODE. The user is expected to provide integer values for the TD\_SEQNO column.

Having a sequence number field is useful if the incoming data has more than one observation/reading at the same timestamp value. For example, consider a PTI table that records the sales at a store on a daily basis. The sales are recorded per day – so for a date of 2017-04-23, there could be 10 sales data. As the time bucket duration is based on DAYS and not timestamp, the time at which a sale is done is not recorded. Adding the sequence number can provide an ordering on the sales for each day.

## Primary Time Index Distribution Strategy

Storage distribution choice	Time interval only (hh:mm:ss)	Time + column list (ID, hh:mm:ss)	Column list only (ID, cost)
In-table logical ordering	Time code only		Time code + sequence number

These are the time series table designer options. Most of it will be easy since they should already understand the data layout and the kinds of queries most commonly needed. Most companies will start with **Time + column list** and **Time code only**. These choices have a dramatic effect on query performance. By distributing data across nodes and AMPs, we ensure parallelism. Furthermore, keys are hashed which provides the top level index without the cost of maintaining B-trees. Then, within the data blocks, there is in-table ordering which sorts the data into the timestamp order. Teradata handles sorting on column lists already so there is no need to organize the data this way when stored on disk.

The rows of a PTI table are distributed based on the TD\_TIMEBUCKET column value and/or one or more column values. Choosing a good distribution strategy depends on the nature of the time series data and the kind of queries that are expected to be used.

**Time interval only** - This kind of distribution is suitable for continuous time series data coming from a single source.

**Time + column list** - This kind of distribution is suitable where there is continuous time series data coming from multiple sources.

**Column list only** - This kind of distribution is suitable where the time series data is short and with or without a logical overlay. For example a finite time series.

## Primary Time Index Tables

### Primary Time Index Configuration Parameters:

```
CREATE [SET|MULTISET] [GLOBAL TEMPORARY | VOLATILE ] TABLE
  series_table_name> [, <table options> ]
  ( [ <generated_column_section>,<column definitions> )
  PRIMARY TIME INDEX <optional_index_name>
    ( <timecode_dt> [, <timezero_date>] [,<timebucket_duration>] [,<columns_clause,>] [, <sequenced_flag> ] )
  [ <as clause> ] [ <index definitions> ] [ <commit options> ] ;
```

<timecode\_dt>: { TIMESTAMP | TIMESTAMP WITH TIME ZONE | DATE }

<timezero\_date>: A DATE value specifying the "time zero" associated with table  
Default timezero\_date is January 1<sup>st</sup>, 1970 @ 00:00:00 hours.

<timebucket\_duration>: A time duration specified by CAL\_YEARS,  
CAL\_MONTHS,CAL\_DAYS,WEEKS, DAYS, HOURS, MINUTES,  
SECONDS, MILLISECONDS, MICROSECONDS.

<columns\_clause,>: COLUMNS ( <column\_list> )

<sequenced\_flag>: {SEQUENCED <optional\_maximum> | NONSEQUENCED }

The configuration parameters associated with the PRIMARY TIME INDEX clause are as follows:

<Timecode\_dt>: { DATE | TIMESTAMP(n) [WITH TIME ZONE] }

This specifies the date-time data type that is used to collect the time series data. The TD\_TIMECODE column that is generated will have the same data type that is specified here and is used to hold the time value associated with the time series data.

<TimeZero\_Date> : DATE

This specifies the earliest date at which the time series data collection starts. If not specified, the default Time Zero date will be set to EPOCH time, January 1st, 1970 @ 00:00:00 hours. Ideally, the Time Zero value should be set to a date just prior to when data collection starts in a table. For example, if a PTI table is created and starts collecting data as of 2017-03-01, then the Time Zero can be set to '2017-01-01'.

<TimeBucket\_Duration> : time\_unit(n) where time\_unit = { CAL\_YEARS | CAL\_MONTHS | CAL\_DAYS | WEEKS | DAYS | HOURS | MINUTES | SECONDS | MILLISECONDS | MICROSECONDS }

A time interval specification that breaks up the time series data into discrete groups called timebuckets. The time units can also be specified using short-hand notation. The short-hand forms are given in Table 1: Short hand forms for time unit durations.

<Columns\_Clause> : COLUMNS(columns\_list)

List of column names that specify the columns to be used to distribute the rows among the AMPs.

<Sequenced\_Flag> : SEQUENCED (max\_val) | NONSEQUENCED

Used to specify an ordering sequence on the time series rows. If the sequenced\_flag is absent or if NONSEQUENCED is specified, the rows are ordered by the TD\_TIMECODE column only. If SEQUENCED flag is specified, the TD\_SEQNO column is added to the table. The user needs to

supply the TD\_SEQNO column values when inserting rows into the table.

## Primary Time Index Tables (cont.)

### Auto-generated columns

#### TD\_TIMEBUCKET

- TD\_TIMEBUCKET BIGINT NOT NULL GENERATED SYSTEM TIMECOLUMN
- Column is generated when the <timebucket\_duration> clause is specified within the PRIMARY TIME INDEX clause
- Values are populated and managed by Teradata
- Hidden column: Cannot be updated, selected or referenced in a query

#### TD\_TIMECODE

- TD\_TIMECODE <timecode\_dt> NOT NULL GENERATED TIMECOLUMN
- Data type should be TIME, TIMESTAMP or DATE – same as the <timecode\_dt> clause within in the PRIMARY TIME INDEX clause
- Value must always be provided by the user

#### TD\_SEQNO

- TD\_SEQNO INT NOT NULL GENERATED TIMECOLUMN
- Generated when the SEQUENCED clause is specified in the PRIMARY TIME INDEX clause
- Used to order the rows in a PTI table along with the TD\_TIMECODE field
- The valid range of TD\_SEQNO is between 1 to 2147483647 inclusively
- Default maximum is 20000

The **TD\_TIMEBUCKET** column is present when a <timebucket\_duration> is specified during table creation. It is a non-null column whose value is populated by Teradata with the value calculated for the time bucket for the row. The column cannot be updated, referenced or selected directly in a query.

The **TD\_TIMECODE** column is always present in a PTI table. This column will contain the time code value at which the measurement/observation occurs. It is a non-null field that is always assigned a value by the user.

The **TD\_SEQNO** field is present when the PTI table is defined as SEQUENCED. The TD\_SEQNO value is used to order the rows of the table along with the TD\_TIMECODE field. It is a non-null, integer field and the value must be supplied by the user. The valid range of a TD\_SEQNO field is 1 to a maximum of 2147483647. If no maximum value is specified in the SEQUENCED(max\_val) specification, then the default maximum is 20000. Note that if the SEQUENCED or NONSEQUENCED flag is not explicitly specified, the table will be non-sequenced by default.

The auto-generated time columns are automatically added to a PTI table definition by Teradata. If the user decides to explicitly specify the auto-generated columns in the CREATE TABLE statement, then *all* the columns applicable for the table must be specified. For example, if the CREATE TABLE statement issued by the user contains the TD\_TIMEBUCKET column only and not the other auto-generated time columns – TD\_TIMECODE

## Time Aware Aggregation Functions

Existing Aggregate Functions	
Average	Count
Describe	Kurtosis
Maximum	Minimum
Percentile	Rank
Skew	Sum
Std. population deviation	Std. sample deviation
Population variance	Sample variance

If not in the list above, then function is not time aware and cannot be used with the GROUP BY TIME clause

New Aggregate Functions	
Bottom	Delta_T
First	Last
Median	Mode
Top	Mean absolute deviation

These new aggregate functions are only invocable with the GROUP BY TIME clause

Note: Group By Time can be used on any table with a time column even if the table does not have a PTI

Here the different time series functions available as part of 16.20 release.

Note: there is an existing MEDIAN function out there ... but it is an Ordered Analytic / Windowed Aggregate function ... not an aggregate function. The ordered analytic function can only be invoked using the ordered analytic syntax. (See Function and Operators SQL manual)

A set of aggregate functions is provided to support time series data (optionally stored in Primary Time Index (PTI) tables). Additionally, some traditional functions support time series as well. To operate on time series data, both time series-specific functions and traditional functions are invoked in a GROUP BY TIME clause.

You can use the following aggregate functions on time series data in PTI tables by using the GROUP BY TIME clause and in non-PTI tables by using the GROUP BY TIME clause with the USING TIMECODE option:

- AVERAGE (AVG)
- COUNT
- KURTOSIS
- MAXIMUM (MAX)
- MINIMUM (MIN)
- RANK (ANSI)
- SKEW
- STANDARD DEVIATION OF A POPULATION (STDDEV\_POP)
- STANDARD DEVIATION OF A SAMPLE (STDDEV\_SAMP)
- SUM
- VARIANCE OF A POPULATION (VAR\_POP)
- VARIANCE OF A SAMPLE (VAR\_SAMP)

## Time Aware Aggregate Example

“For each beacon sensor location, show me the total foot traffic in a ½ hour increment, over 2 hours”



```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME, BEACON_ID, SUM(TRAFFIC) FROM BEACONS
WHERE DATE_TIME BETWEEN TIMESTAMP '2017-08-11 08:00:00' AND TIMESTAMP '2017-08-11 10:00:00'
GROUP BY TIME( MINUTES(30) AND BEACON_ID) USING TIMECODE(Date_Time)
ORDER BY BEACON_ID, $TD_GROUP_BY_TIME;
```

TIMECODE-RANGE	GROUP_BY_#	BEACON	TRAFFIC
'2017-08-11 08:00:00', '2017-08-11 08:30:00'	1	22	50
'2017-08-11 08:30:00', '2017-08-11 09:00:00'	2	22	95
'2017-08-11 09:00:00', '2017-08-11 09:30:00'	3	22	114
'2017-08-11 09:30:00', '2017-08-11 10:00:00'	4	22	37
'2017-08-11 08:00:00', '2017-08-11 08:30:00'	1	23	80
'2017-08-11 08:30:00', '2017-08-11 09:00:00'	2	23	65
'2017-08-11 09:30:00', '2017-08-11 10:00:00'	4	23	40

When our user wants to understand traffic over time, it gets a bit more complicated to write the query. Typically, there is a lot of time arithmetic involved and it's cumbersome to change once coded.

By using the GROUP BY TIME function, this is all resolved. Users can now easily ask their questions and then quickly iterate if a different group is necessary.

Did you see what really happened here? The data scientist just went from days of work down to minutes. Organizing the data set manually and applying any kind of analytic function is not simply expressing the SQL and attaching functions. All that data preparation we hear so many complaints about just vanished because the database solved most everything during ingest and optimized the access.

This query would be difficult if not for the time series features. Specifically, it would end up as multiple nested SQL statements. As the user wants to iterate and change the granularity or then join to other tables with different time granularity, it becomes much more complex. Having GROUP BY TIME resolves these complexity and allows users to do what they need to do: Analytics!

## Time Series – FILL Clause

FILL SCHEME	AGGREGATE RESULT
NULLS	Null
<Constant>	A constant value
PREV/PREVIOUS	Same as the previous time bucket's result
NEXT	Same as the next time bucket's result

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME, BEACON_ID, SUM(TRAFFIC) FROM BEACONS
WHERE DATE_TIME BETWEEN TIMESTAMP '2017-08-11 08:00:00'
AND TIMESTAMP '2017-08-11 10:00:00'
GROUP BY TIME( MINUTES(30) AND BEACON_ID)
USING TIMECODE(DATE_TIME)
FILL (NULLS)
ORDER BY BEACON_ID, $TD_GROUP_BY_TIME;
```

Use the FILL clause to replace missing values with a constant value for time buckets with missing values.

## Many SQL Table Designs Include Time

	Partitioned Primary Index (PPI)	Temporal Tables	Primary Time Index (PTI)
Business	<ul style="list-style-type: none"> <li>Multi-dimensional analytics</li> <li>Hierarchical analytics</li> <li>Date, character, or numeric levels</li> </ul>	<ul style="list-style-type: none"> <li>Time periods (ranges)</li> <li>Historical relevance</li> <li>Audit – what was the situation when...</li> </ul>	<ul style="list-style-type: none"> <li>High volume time stamped data</li> <li>Time aware analytics</li> <li>Sorted data</li> <li>Unique algorithms</li> </ul>
Technology	<ul style="list-style-type: none"> <li>Multi-level (up to 64)</li> <li>Does not effect row distribution to the AMPs</li> <li>Data is not ordered</li> </ul>	<ul style="list-style-type: none"> <li>Slowly changing dimensions</li> <li>Insert, update, delete</li> <li>Normalize and overlap functions</li> </ul>	<ul style="list-style-type: none"> <li>Distribution to AMPs by time buckets</li> <li>Updates/deletes rare</li> <li>Insert late arrival data</li> <li>Multivariate payload common</li> </ul>

All table types can use "GROUP BY TIME"

**PPI** organizes data within the AMP. It does not determine which AMP the data goes to. This yields highly effective all AMP operations.

It also has multi level partitions, which uses different keys within the partition to segment further. This helps with BI tools. PPI and MLPPI helps get rid of OLAP cubes.

**Temporal** is a time based table.

This is about effective management of a time period. When is a row effective within a **time period**. And when did the RDBMS know about this row.

When did I know this happened? When did I know what? Very useful in audits. Nothing here affects row redistribution.

We are recording when changes happen to a row.

Normalize and overlap are functions for this.

**PTI**

Now its about the buckets. If I make the primary index time, all the events would be on different AMPs. This would cause massive redistributions. So the buckets collect events on an AMP and collocating records that are grouped in the bucket

You actually can do updates and deletes. But should not be doing many of them. Rarely should you be updating the sensor reading.

It doesn't really append, it just creates another partition on another AMP

If data arrives late, data is still stored in the right bucket.

The common SQL functions –bottom, top, median, variance, etc. – are available on all tables.

## Analytics in Action

“How Can We Serve our High-Value Customers Better?”

### Business data

- High-value customers
- Sales/order history



“Identify high-value customers whose purchase interests match our current product promotions and currently live near one of our stores. Send an e-mail with a special discount coupon.”

### Customer locations (Geospatial)

- Current customers' addresses and distance to our store(s) and competitors
- Beacon data

### Customer interests and product availability (Time-based)

- Customers recently and frequently searched/browsed products
- Current product sell rate and inventory at our store(s)

### Valid promotion periods and seasonality (Temporal)

- Store hours
- Current promotion begin/end dates
- Holidays and special events

<https://web.microsoftstream.com/video/3f8dd0d6-694b-4230-9edb-e3ad8cdc166c?list=studio>

Let's look at this 4D Analytics in action using a theoretical example of a retailer with existing promotions that wants to attract high-value customers into the store.

As a strategy, the company wants to identify high-value customers whose purchase interests match current product promotions and are physically near the store, then send an e-mail with a special discount coupon. To do this, the company can integrate:

- 1) Current customer location. For example, it may identify potential customers who are currently within a 1-mile radius from the store. This is 'where' analysis.
- 2) Particular product interests of the customers based on their recent and frequent web searches or browsing data, as well as the real-time or near-real time sell-rate and inventory level of products. This usually is time-based analysis.
- 3) There are particular begin and end dates associated with promotions, and even store hours. Also, some product sales may be highly affected by holidays, seasonality, or special events going on in the area. This usually is temporal.
- 4) All of those can be combined with the business data, such as a pre-existing list of high-value customers and their sales and order history.

As you can see, integration of the business and operational analytics with the analytics capabilities for 'where' and 'when' can provide powerful insights that can create actions and improve business outcomes.

# Time Series

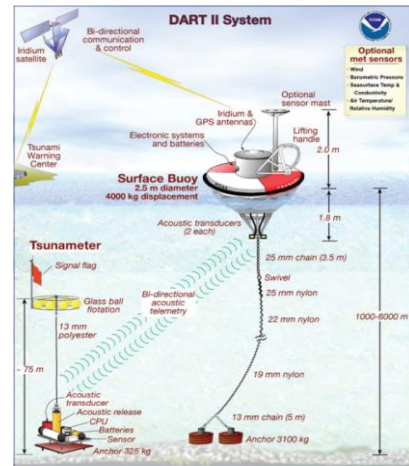
This section will cover time series.

## Time Series

**Time Series data** is data that is continuously produced and collected over a period of time

Time Series features in Vantage allows the user to capture and store Time Series data:

- Time Series data can be stored in tables with a **Primary Time Index** defined on them
- A PTI table is 'time-series' aware and provides different ways to **store and order** the time series data
- Optimized for **time range** queries
- The feature also supports **time-aware aggregate operations** using the GROUP BY TIME clause
- Useful for forecasting, detecting patterns and trends, risk reduction, etc.



Time Series data is data that is continuously produced and collected over a period of time. This kind of data is typically generated by machines such as sensors and other applications and devices that make up the Internet Of Things (IOT). Each data point in the Time Series data set is associated with a timestamp and an observed value at that time. Time Series data can be stored and analyzed to provide capabilities such as forecasting, detecting patterns and trends, anomaly detection, risk reduction etc.

The Teradata Time Series feature introduced in Teradata release 16.20 allows the user to capture and store Time Series data and perform useful aggregate operations and analytics on the data. Time Series data can be stored in tables with a new construct Primary Time Index defined on them. A Primary Time Index table is 'time-series' aware and provides different ways to store and order the time series data. The feature also supports time-aware aggregate operations that can be performed on the data set. This is done using the GROUP BY TIME clause and a set of 'time-aware' aggregate functions. All this can be combined with the existing Teradata database capabilities such as a full range of SQL support, rich collection of native and complex data types including JSON, AVRO, CSV, XML and wide range of load and extract utilities resulting in a powerful, feature-rich Time Series database offering.

## Time Series Categories

### Class I: The 7/24 infinite time series

- Hydrology: USGS uses river monitoring devices to collect time series data on all major rivers and streams
- Oceanography: There is a world-wide buoy system collecting data on a 7/24 basis
- Building Monitoring systems, Manufacturing Line Monitoring Systems

### Class II: Time Series with a “logical overlay”

- Automobile “Trip” – Start-Engine; drive from location A to location B; shut-down engine
- Plane “Flight” – Start-Engine; Take-off; Fly from location A to location B; Land; Shut-down engine
- Cargo Ship “Voyage” – Start Engine; Raise Anchor; Navigate from A to B; Drop Anchor; Shut down engine

### Class III: Fixed-size (few thousand entries) Scientific Trace Time Series

- Oil Exploration: Seismic Traces used to determine geographical sub-layers
- Medicine: Traces associated with an Ultrasound Scan or CAT Scan
- Scientific: Traces associated with the Electron Microscope to investigate crystal or cell structures

There are three categories of Time-series data.

#### Class I

Class I is 7/24 infinite time-series. In this class, data continuously collected, 7 days a week, 24 hour a day, 365 days a year...nonstop. For example, the United States Geological Survey (USGS) has Buoys in major rivers and streams collecting on a 7/24 basis and for each Buoy station, you would have a time series you keep adding data to the end. This would keep going on and on collecting until infinity.

#### Class II

With Class II data, it's a logical overlay... think about automobile... For a given trip, you start the engine and drive from point A to a destination B and then turn off the car. Overlay this trip data on the data that is collected for a given car for say #113 or air plane # 933... The importance of these trip overlays is for analysis where you may want to compare one trip to another trip for this same car #113 or compare one air plane type to another plane that does the same trip say Chicago to San Diego.

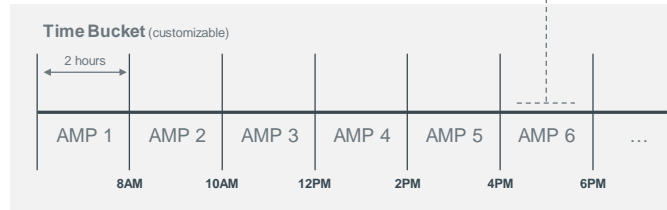
#### Class III

Class III is fixed size...Seismic Traces used to determine geographical sub-layers where the receiving electronics are setup for this discrete series. For example, data may be collected every millisecond for 6 seconds or 6,000 entries. That is the complete series... then this fixed data represents a fixed size of time series data to analyze. Other fields may have time-series data just long enough to collect an ultra-scan, Cat scan or Microscope reading to be analyzed later. No additional data is added to the end.

## Primary Time Index (PTI)

### High Performance Parallelism with Efficient Storage and Access

- High performance parallel distribution
- Customizable duration of Time Bucket provides control of distribution
- Fast Primary AMP access
- Customizable distribution provides AMP-local processing minimizes data movement and speeds query processing
- Fully automated; set once



### Three Storage Distribution Choices

Time Interval  
only

Time Interval  
and Column List

Column List only

A PTI table can be defined with 3 different data distribution strategies and 2 different ordering methods. We will discuss the distribution strategies in the next slide.

For ordering strategy, a PTI table can be defined as SEQUENCED or NONSEQUENCED in the PRIMARY TIME INDEX clause.

A PTI table is NONSEQUENCED by default i.e. if the NONSEQUENCED keyword is omitted from the PRIMARY TIME INDEX clause, the table will be considered to be non-sequenced.

A sequenced PTI table will have the auto-generated TD\_SEQNO column. The rows in the table are first ordered by TD\_TIMECODE and then by the TD\_SEQNO value. When more than one row has the same TD\_TIMECODE value, those rows are ordered by the TD\_SEQNO value within the same TD\_TIMECODE. The user is expected to provide integer values for the TD\_SEQNO column.

Having a sequence number field is useful if the incoming data has more than one observation/reading at the same timestamp value. For example, consider a PTI table that records the sales at a store on a daily basis. The sales are recorded per day – so for a date of 2017-04-23, there could be 10 sales data. As the time bucket duration is based on DAYS and not timestamp, the time at which a sale is done is not recorded. Adding the sequence number can provide an ordering on the sales for each day.

## Primary Time Index Distribution Strategy

Storage distribution choice	Time interval only (hh:mm:ss)	Time + column list (ID, hh:mm:ss)	Column list only (ID, cost)
In-table logical ordering	Time code only	Time code + sequence number	

These are the time series table designer options. Most of it will be easy since they should already understand the data layout and the kinds of queries most commonly needed. Most companies will start with **Time + column list** and **Time code only**. These choices have a dramatic effect on query performance. By distributing data across nodes and AMPs, we ensure parallelism. Furthermore, keys are hashed which provides the top level index without the cost of maintaining B-trees. Then, within the data blocks, there is in-table ordering which sorts the data into the timestamp order. Teradata handles sorting on column lists already so there is no need to organize the data this way when stored on disk.

The rows of a PTI table are distributed based on the TD\_TIMEBUCKET column value and/or one or more column values. Choosing a good distribution strategy depends on the nature of the time series data and the kind of queries that are expected to be used.

**Time interval only** - This kind of distribution is suitable for continuous time series data coming from a single source.

**Time + column list** - This kind of distribution is suitable where there is continuous time series data coming from multiple sources.

**Column list only** - This kind of distribution is suitable where the time series data is short and with or without a logical overlay. For example a finite time series.

## Primary Time Index Tables

### Primary Time Index Configuration Parameters:

```
CREATE [SET|MULTISET] [GLOBAL TEMPORARY | VOLATILE ] TABLE
  series_table_name> [, <table options> ]
  ( [ <generated_column_section>,<column definitions> )
  PRIMARY TIME INDEX <optional_index_name>
    ( <timecode_dt> [, <timezero_date>] [,<timebucket_duration>] [,<columns_clause,>] [, <sequenced_flag> ] )
  [ <as clause> ] [ <index definitions> ] [ <commit options> ] ;
```

<timecode\_dt>: { TIMESTAMP | TIMESTAMP WITH TIME ZONE | DATE }

<timezero\_date>: A DATE value specifying the "time zero" associated with table  
Default timezero\_date is January 1<sup>st</sup>, 1970 @ 00:00:00 hours.

<timebucket\_duration>: A time duration specified by CAL\_YEARS,  
CAL\_MONTHS,CAL\_DAYS,WEEKS, DAYS, HOURS, MINUTES,  
SECONDS, MILLISECONDS, MICROSECONDS.

<columns\_clause,>: COLUMNS ( <column\_list> )

<sequenced\_flag>: {SEQUENCED <optional\_maximum> | NONSEQUENCED }

The configuration parameters associated with the PRIMARY TIME INDEX clause are as follows:

<Timecode\_dt>: { DATE | TIMESTAMP(n) [WITH TIME ZONE] }

This specifies the date-time data type that is used to collect the time series data. The TD\_TIMECODE column that is generated will have the same data type that is specified here and is used to hold the time value associated with the time series data.

<TimeZero\_Date> : DATE

This specifies the earliest date at which the time series data collection starts. If not specified, the default Time Zero date will be set to EPOCH time, January 1st, 1970 @ 00:00:00 hours. Ideally, the Time Zero value should be set to a date just prior to when data collection starts in a table. For example, if a PTI table is created and starts collecting data as of 2017-03-01, then the Time Zero can be set to '2017-01-01'.

<TimeBucket\_Duration> : time\_unit(n) where time\_unit = { CAL\_YEARS | CAL\_MONTHS | CAL\_DAYS | WEEKS | DAYS | HOURS | MINUTES | SECONDS | MILLISECONDS | MICROSECONDS }

A time interval specification that breaks up the time series data into discrete groups called timebuckets. The time units can also be specified using short-hand notation. The short-hand forms are given in Table 1: Short hand forms for time unit durations.

<Columns\_Clause> : COLUMNS(columns\_list)

List of column names that specify the columns to be used to distribute the rows among the AMPs.

<Sequenced\_Flag> : SEQUENCED (max\_val) | NONSEQUENCED

Used to specify an ordering sequence on the time series rows. If the sequenced\_flag is absent or if NONSEQUENCED is specified, the rows are ordered by the TD\_TIMECODE column only. If SEQUENCED flag is specified, the TD\_SEQNO column is added to the table. The user needs to

supply the TD\_SEQNO column values when inserting rows into the table.

## Primary Time Index Tables (cont.)

### Auto-generated columns

#### TD\_TIMEBUCKET

- TD\_TIMEBUCKET BIGINT NOT NULL GENERATED SYSTEM TIMECOLUMN
- Column is generated when the <timebucket\_duration> clause is specified within the PRIMARY TIME INDEX clause
- Values are populated and managed by Teradata
- Hidden column: Cannot be updated, selected or referenced in a query

#### TD\_TIMECODE

- TD\_TIMECODE <timecode\_dt> NOT NULL GENERATED TIMECOLUMN
- Data type should be TIME, TIMESTAMP or DATE – same as the <timecode\_dt> clause within in the PRIMARY TIME INDEX clause
- Value must always be provided by the user

#### TD\_SEQNO

- TD\_SEQNO INT NOT NULL GENERATED TIMECOLUMN
- Generated when the SEQUENCED clause is specified in the PRIMARY TIME INDEX clause
- Used to order the rows in a PTI table along with the TD\_TIMECODE field
- The valid range of TD\_SEQNO is between 1 to 2147483647 inclusively
- Default maximum is 20000

The **TD\_TIMEBUCKET** column is present when a <timebucket\_duration> is specified during table creation. It is a non-null column whose value is populated by Teradata with the value calculated for the time bucket for the row. The column cannot be updated, referenced or selected directly in a query.

The **TD\_TIMECODE** column is always present in a PTI table. This column will contain the time code value at which the measurement/observation occurs. It is a non-null field that is always assigned a value by the user.

The **TD\_SEQNO** field is present when the PTI table is defined as SEQUENCED. The TD\_SEQNO value is used to order the rows of the table along with the TD\_TIMECODE field. It is a non-null, integer field and the value must be supplied by the user. The valid range of a TD\_SEQNO field is 1 to a maximum of 2147483647. If no maximum value is specified in the SEQUENCED(max\_val) specification, then the default maximum is 20000. Note that if the SEQUENCED or NONSEQUENCED flag is not explicitly specified, the table will be non-sequenced by default.

The auto-generated time columns are automatically added to a PTI table definition by Teradata. If the user decides to explicitly specify the auto-generated columns in the CREATE TABLE statement, then *all* the columns applicable for the table must be specified. For example, if the CREATE TABLE statement issued by the user contains the TD\_TIMEBUCKET column only and not the other auto-generated time columns – TD\_TIMECODE

## Time Aware Aggregation Functions

Existing Aggregate Functions	
Average	Count
Describe	Kurtosis
Maximum	Minimum
Percentile	Rank
Skew	Sum
Std. population deviation	Std. sample deviation
Population variance	Sample variance

If not in the list above, then function is not time aware and cannot be used with the GROUP BY TIME clause

New Aggregate Functions	
Bottom	Delta_T
First	Last
Median	Mode
Top	Mean absolute deviation

These new aggregate functions are only invocable with the GROUP BY TIME clause

Note: Group By Time can be used on any table with a time column even if the table does not have a PTI

Here the different time series functions available as part of 16.20 release.

Note: there is an existing MEDIAN function out there ... but it is an Ordered Analytic / Windowed Aggregate function ... not an aggregate function. The ordered analytic function can only be invoked using the ordered analytic syntax. (See Function and Operators SQL manual)

A set of aggregate functions is provided to support time series data (optionally stored in Primary Time Index (PTI) tables). Additionally, some traditional functions support time series as well. To operate on time series data, both time series-specific functions and traditional functions are invoked in a GROUP BY TIME clause.

You can use the following aggregate functions on time series data in PTI tables by using the GROUP BY TIME clause and in non-PTI tables by using the GROUP BY TIME clause with the USING TIMECODE option:

- AVERAGE (AVG)
- COUNT
- KURTOSIS
- MAXIMUM (MAX)
- MINIMUM (MIN)
- RANK (ANSI)
- SKEW
- STANDARD DEVIATION OF A POPULATION (STDDEV\_POP)
- STANDARD DEVIATION OF A SAMPLE (STDDEV\_SAMP)
- SUM
- VARIANCE OF A POPULATION (VAR\_POP)
- VARIANCE OF A SAMPLE (VAR\_SAMP)

## Time Aware Aggregate Example

“For each beacon sensor location, show me the total foot traffic in a ½ hour increment, over 2 hours”



```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME, BEACON_ID, SUM(TRAFFIC) FROM BEACONS
WHERE DATE_TIME BETWEEN TIMESTAMP '2017-08-11 08:00:00' AND TIMESTAMP '2017-08-11 10:00:00'
GROUP BY TIME( MINUTES(30) AND BEACON_ID) USING TIMECODE(Date_Time)
ORDER BY BEACON_ID, $TD_GROUP_BY_TIME;
```

TIMECODE-RANGE	GROUP_BY_#	BEACON	TRAFFIC
'2017-08-11 08:00:00', '2017-08-11 08:30:00'	1	22	50
'2017-08-11 08:30:00', '2017-08-11 09:00:00'	2	22	95
'2017-08-11 09:00:00', '2017-08-11 09:30:00'	3	22	114
'2017-08-11 09:30:00', '2017-08-11 10:00:00'	4	22	37
'2017-08-11 08:00:00', '2017-08-11 08:30:00'	1	23	80
'2017-08-11 08:30:00', '2017-08-11 09:00:00'	2	23	65
'2017-08-11 09:30:00', '2017-08-11 10:00:00'	4	23	40

When our user wants to understand traffic over time, it gets a bit more complicated to write the query. Typically, there is a lot of time arithmetic involved and it's cumbersome to change once coded.

By using the GROUP BY TIME function, this is all resolved. Users can now easily ask their questions and then quickly iterate if a different group is necessary.

Did you see what really happened here? The data scientist just went from days of work down to minutes. Organizing the data set manually and applying any kind of analytic function is not simply expressing the SQL and attaching functions. All that data preparation we hear so many complaints about just vanished because the database solved most everything during ingest and optimized the access.

This query would be difficult if not for the time series features. Specifically, it would end up as multiple nested SQL statements. As the user wants to iterate and change the granularity or then join to other tables with different time granularity, it becomes much more complex. Having GROUP BY TIME resolves these complexity and allows users to do what they need to do: Analytics!

## Time Series – FILL Clause

FILL SCHEME	AGGREGATE RESULT
NULLS	Null
<Constant>	A constant value
PREV/PREVIOUS	Same as the previous time bucket's result
NEXT	Same as the next time bucket's result

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME, BEACON_ID, SUM(TRAFFIC) FROM BEACONS
WHERE DATE_TIME BETWEEN TIMESTAMP '2017-08-11 08:00:00'
AND TIMESTAMP '2017-08-11 10:00:00'
GROUP BY TIME( MINUTES(30) AND BEACON_ID)
USING TIMECODE(DATE_TIME)
FILL (NULLS)
ORDER BY BEACON_ID, $TD_GROUP_BY_TIME;
```

Use the FILL clause to replace missing values with a constant value for time buckets with missing values.

## Many SQL Table Designs Include Time

	Partitioned Primary Index (PPI)	Temporal Tables	Primary Time Index (PTI)
Business	<ul style="list-style-type: none"> <li>Multi-dimensional analytics</li> <li>Hierarchical analytics</li> <li>Date, character, or numeric levels</li> </ul>	<ul style="list-style-type: none"> <li>Time periods (ranges)</li> <li>Historical relevance</li> <li>Audit – what was the situation when...</li> </ul>	<ul style="list-style-type: none"> <li>High volume time stamped data</li> <li>Time aware analytics</li> <li>Sorted data</li> <li>Unique algorithms</li> </ul>
Technology	<ul style="list-style-type: none"> <li>Multi-level (up to 64)</li> <li>Does not effect row distribution to the AMPs</li> <li>Data is not ordered</li> </ul>	<ul style="list-style-type: none"> <li>Slowly changing dimensions</li> <li>Insert, update, delete</li> <li>Normalize and overlap functions</li> </ul>	<ul style="list-style-type: none"> <li>Distribution to AMPs by time buckets</li> <li>Updates/deletes rare</li> <li>Insert late arrival data</li> <li>Multivariate payload common</li> </ul>

All table types can use "GROUP BY TIME"

**PPI** organizes data within the AMP. It does not determine which AMP the data goes to. This yields highly effective all AMP operations.

It also has multi level partitions, which uses different keys within the partition to segment further. This helps with BI tools. PPI and MLPPI helps get rid of OLAP cubes.

**Temporal** is a time based table.

This is about effective management of a time period. When is a row effective within a **time period**. And when did the RDBMS know about this row.

When did I know this happened? When did I know what? Very useful in audits. Nothing here affects row redistribution.

We are recording when changes happen to a row.

Normalize and overlap are functions for this.

**PTI**

Now its about the buckets. If I make the primary index time, all the events would be on different AMPs. This would cause massive redistributions. So the buckets collect events on an AMP and collocating records that are grouped in the bucket

You actually can do updates and deletes. But should not be doing many of them. Rarely should you be updating the sensor reading.

It doesn't really append, it just creates another partition on another AMP

If data arrives late, data is still stored in the right bucket.

The common SQL functions –bottom, top, median, variance, etc. – are available on all tables.

## Analytics in Action

“How Can We Serve our High-Value Customers Better?”

### Business data

- High-value customers
- Sales/order history



“Identify high-value customers whose purchase interests match our current product promotions and currently live near one of our stores. Send an e-mail with a special discount coupon.”

### Customer locations (Geospatial)

- Current customers' addresses and distance to our store(s) and competitors
- Beacon data

### Customer interests and product availability (Time-based)

- Customers recently and frequently searched/browsed products
- Current product sell rate and inventory at our store(s)

### Valid promotion periods and seasonality (Temporal)

- Store hours
- Current promotion begin/end dates
- Holidays and special events

<https://web.microsoftstream.com/video/3f8dd0d6-694b-4230-9edb-e3ad8cdc166c?list=studio>

Let's look at this 4D Analytics in action using a theoretical example of a retailer with existing promotions that wants to attract high-value customers into the store.

As a strategy, the company wants to identify high-value customers whose purchase interests match current product promotions and are physically near the store, then send an e-mail with a special discount coupon. To do this, the company can integrate:

- 1) Current customer location. For example, it may identify potential customers who are currently within a 1-mile radius from the store. This is 'where' analysis.
- 2) Particular product interests of the customers based on their recent and frequent web searches or browsing data, as well as the real-time or near-real time sell-rate and inventory level of products. This usually is time-based analysis.
- 3) There are particular begin and end dates associated with promotions, and even store hours. Also, some product sales may be highly affected by holidays, seasonality, or special events going on in the area. This usually is temporal.
- 4) All of those can be combined with the business data, such as a pre-existing list of high-value customers and their sales and order history.

As you can see, integration of the business and operational analytics with the analytics capabilities for 'where' and 'when' can provide powerful insights that can create actions and improve business outcomes.

# Aggregate Functions

## Overview

The following sections describe SQL aggregate functions.

For information on:

- Window aggregate functions and their Teradata-specific equivalents, see [Window Aggregate Functions](#).
- Aggregate user-defined functions (UDFs), see "Aggregate UDF" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210.
- Window aggregate UDFs, see "Window Aggregate UDF" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210.

## About Aggregate Functions

Aggregate functions are typically used in arithmetic expressions. Aggregate functions operate on a group of rows and return a single numeric value in the result table for each group.

In the following statement, the SUM aggregate function operates on the group of rows defined by the Sales\_Table table:

```
SELECT SUM(Total_Sales)
FROM Sales_Table;
Sum(Total_Sales)
-----
          5192.40
```

You can use GROUP BY clauses to produce more complex, finer grained results in multiple result values. In the following statement, the SUM aggregate function operates on groups of rows defined by the Product\_ID column in the Sales\_Table table:

```
SELECT Product_ID, SUM(Total_Sales)
FROM Sales_Table
GROUP BY Product_ID;
Product_ID  Sum(Total_Sales)
-----
          101          2100.00
          107          1000.40
          102          2092.00
```

## Aggregates in the Select List

Aggregate functions are normally used in the expression list of a SELECT statement and in the summary list of a WITH clause.

## Aggregates and GROUP BY

If you use an aggregate function in the select list of an SQL statement, then either all other columns occurring in the select list must also be referenced by means of aggregate functions or their column name must appear in a GROUP BY clause. For example, the following statement uses an aggregate function and a column in the select list and references the column name in the GROUP BY clause:

```
SELECT COUNT(*), Product_ID
FROM Sales_Table
GROUP BY Product_ID;
```

The reason for this is that aggregates return only one value, while a non-GROUP BY column reference can return any number of values.

## Aggregates and Date

It is valid to apply AVG, MIN, MAX, or COUNT to a date. It is not valid to specify SUM(date).

## Aggregates and Literal Expressions in the Select List

Literal expressions in the select list may optionally appear in the GROUP BY clause. For example, the following statement uses an aggregate function and a literal expression in the select list, and does not use a GROUP BY clause:

```
SELECT COUNT(*),
SUBSTRING( CAST( CURRENT_TIME(0) AS CHAR(14) ) FROM 1 FOR 8 )
FROM Sales_Table;
```

The results of such statements when the table has no rows depends on the type of literal expression.

IF the literal expression ...	THEN the result of the literal expression in the query result is ...
does not contain a column reference is a non-deterministic function, such as RANDOM	the value of the literal expression. Functions such as RANDOM are computed in the immediate retrieve step of the request instead of in the aggregation step. Here is an example: SELECT COUNT(*), SUBSTRING(CAST(CURRENT_TIME(0) AS CHAR(14)) FROM 1 FOR 8) FROM Sales_Table;

IF the literal expression ...	THEN the result of the literal expression in the query result is ...
	Count(*) Substring(Current Time(0) From 1 For 8) ----- 0 09:01:43
contains a column reference is a UDF	NULL. Here is an example: SELECT COUNT(*), UDF_CALC(1,2) FROM Sales_Table; Count(*) UDF_CALC(1,2) ----- 0          ?

## Nesting Aggregates

Aggregate operations cannot be nested. The following aggregate is not valid and returns an error:

```
AVG(MAXIMUM (Salary))
```

Although direct nesting of aggregates is not supported, nested aggregates can be evaluated using a derived table that contains the aggregates to be nested. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

Also, aggregates can be nested in aggregate window functions. The following statement is valid and includes an aggregate SUM function nested in a RANK window function:

```
SELECT region
       ,product
       ,SUM(amount)
       ,RANK() OVER (PARTITION BY region ORDER by SUM (amount))
FROM table;
```

## Results of Aggregation on Zero Rows

Aggregation on zero rows behaves as indicated by the following table.

This form of aggregate function ...	Returns this result when there are zero rows ...
COUNT(expression) WHERE ...	0
all other forms of <i>aggregate_operator (expression)</i> WHERE ...	Null
<i>aggregate_operator (expression)</i> ... GROUP BY ... <i>aggregate_operator (expression)</i> ... HAVING ...	No Record Found

## Aggregates and Nulls

Aggregates (with the exception of COUNT(\*)) ignore nulls in all computations.

### Note:

A UDT column value is null only when you explicitly place a NULL in a column, not when a UDT instance has an attribute that is set to null.

Ignoring nulls can result in apparent nontransitive anomalies. For example, if there are nulls in either column A or column B (or both), then the following expression is virtually always true.

$$\text{SUM}(A) + \text{SUM}(B) <> \text{SUM}(A+B)$$

The only exception to this is the case in which the values for columns A and B are both null in the same rows, because in those cases the entire row is disregarded in the aggregation. This is a trivial case that does not violate the general rule.

More formally stated, if and only if field A and field B are both null for every occurrence of a null in either field is the above inequality false.

For examples that illustrate this behavior, see "Example: Employees Returned as Nulls" and "Example: Counting Employees Not Yet Assigned to a Department" in [Result Type and Attributes](#). Note that the aggregates are behaving exactly as they should, the results are not mathematically anomalous.

There are several ways to work around this apparent nontransitivity issue if it presents a problem. Either solution provides the same consistent results.

- Always define your numeric columns as NOT NULL DEFAULT 0.
- Use the ZEROIFNULL function within the aggregate function to convert any nulls to zeros for the computation, for example SUM(ZEROIFNULL(x) + ZEROIFNULL(y)), which produces the same result as SUM(ZEROIFNULL(x)) + SUM(ZEROIFNULL(y)).

## Aggregate Operations on Floating Point Data

Operations involving floating point numbers are not always associative due to approximation and rounding errors: ((A + B) + C) is not always equal to (A + (B + C)).

Although not readily apparent, the non-associativity of floating point arithmetic can also affect aggregate operations: you can get different results each time you use an aggregate function on a given set of floating point data. When Teradata Database performs an aggregation, it accumulates individual terms from each AMP involved in the computation and evaluates the terms in order of arrival to produce the final result. Because the order of evaluation can produce slightly different results, and because the order in which individual AMPs finish their part of the work is unpredictable, the results of an aggregate function on the same data on the same system can vary.

## Aggregates and LOBs

Aggregates do not operate on CLOB or BLOB data types.

## Aggregates and Period Data Types

Aggregates (with the exception of COUNT) do not operate on Period data types.

## Aggregates and SELECT AND CONSUME Statements

Aggregates cannot appear in SELECT AND CONSUME statements.

## Aggregates and Recursive Queries

Aggregate functions cannot appear in a recursive statement of a recursive query. However, a non-recursive seed statement in a recursive query can specify an aggregate function.

## Aggregates in WHERE and HAVING Clauses

Aggregates can appear in the following types of clauses:

- The WHERE clause of an ABORT statement to specify an abort condition.  
But an aggregate function cannot appear in the WHERE clause of a SELECT statement.
- A HAVING clause to specify a group condition.

## DISTINCT Option

The DISTINCT option specifies that duplicate values are not to be used when an expression is processed.

The following SELECT returns the number of unique job titles in a table.

```
SELECT COUNT(DISTINCT JobTitle) FROM Employee;
```

A query can have multiple aggregate functions that use DISTINCT with the same expression, as shown by the following example.

```
SELECT SUM(DISTINCT x), AVG(DISTINCT x) FROM XTable;
```

A query can also have multiple aggregate functions that use DISTINCT with different expressions, for example:

```
SELECT SUM(DISTINCT x), SUM(DISTINCT y) FROM XYTable;
```

## Aggregates and Row Level Security Tables

When a request that includes an aggregate function, such as SUM, COUNT, MAX, MIN or AVG, references a table protected by row level security, the aggregation is based on only the rows that are accessible to the requesting user. In order to apply all rows of the table to the aggregation, the user must have one of the following:

- The required security credentials to access all rows of the table.
- The required OVERRIDE privileges on the security constraints in the table.

## Time Series Aggregate Functions Overview

A set of aggregate functions is provided to support time series data (optionally stored in Primary Time Index (PTI) tables). Additionally, some traditional functions support time series as well. To operate on time series data, both time series-specific functions and traditional functions are invoked in a GROUP BY TIME clause.

## Traditional Aggregate Functions that Support Time Series

You can use the following aggregate functions on time series data in PTI tables by using the GROUP BY TIME clause and in non-PTI tables by using the GROUP BY TIME clause with the USING TIMECODE option. For more information on these functions, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

- AVERAGE
- COUNT
- KURTOSIS
- MAXIMUM
- MINIMUM
- RANK (ANSI)
- SKEW
- STANDARD DEVIATION OF A POPULATION (STDDEV\_POP)
- STANDARD DEVIATION OF A SAMPLE (STDDEV\_SAMP)
- SUM
- VARIANCE OF A POPULATION (VAR\_POP)
- VARIANCE OF A SAMPLE (VAR\_SAMP)

## Related Topics

For more information on potential problems associated with floating point values in computations, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

For more details on:

- Window aggregate functions and their Teradata-specific equivalents, see [Window Aggregate Functions](#).
- Aggregate user-defined functions (UDFs), see *Teradata Vantage™ NewSQL Engine Security Administration*, B035-1100.
- Window aggregate UDFs, see "Window Aggregate UDF" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210.
- Row level security, see *Teradata Vantage™ NewSQL Engine Security Administration*, B035-1100.
- Time series-specific aggregate functions, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## AVG

### Purpose

Returns the arithmetic average of all values in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

— AVERAGE — ( — DISTINCT — value_expression ) —
— AVG —
— AVE —
— ALL —

```

## Syntax Elements

### ALL

All values that are not null of *value\_expression*, including duplicates, are included in the computation.

### DISTINCT

Exclude duplicates specified by *value\_expression* from the computation.

### *value\_expression*

A literal or column expression for which an average is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

AVERAGE and AVE are Teradata extensions to the ANSI standard.

## Return Value

This function returns the REAL data type.

## Computation of INTEGER or DECIMAL Values

An AVG of a DECIMAL or INTEGER value may overflow if the individual values are very large or if there is a large number of values.

If this occurs, change the AVG call to include a CAST function that converts the DECIMAL or INTEGER values to REAL as shown in the following example:

```
AVG(CAST(value AS REAL) )
```

Casting the values as REAL before averaging causes a slight loss in precision.

The type of the result is REAL in either case, so the only effect of the CAST is to accept a slight loss of precision where a result might not otherwise be available at all.

If x is an integer, AVG does not display a fractional value. A fractional value may be obtained by casting the value as DECIMAL, for example the following CAST to DECIMAL.

```
CAST(AVG(value) AS DECIMAL(9,2))
```

## Restrictions

AVG is valid only for numeric data.

Nulls are not included in the result computation.

## Example: Using the AVG Function

### Example: Querying the Sales Table for Average Sales by Region

This example queries the sales table for average sales by region and returns the following results.

```
SELECT Region, AVG(sales)
FROM sales_tbl
GROUP BY Region
ORDER BY Region;

Region  Average (sales)
```

-----	-----
North	21840.17
East	55061.32
Midwest	15535.73

For time series examples, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## AVG Window Function

For the AVG window function that computes a group, cumulative, or moving average, see [Window Aggregate Functions](#).

## Related Topics

For more information, see:

- For more information on potential problems associated with floating point values in computations, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- *Teradata Vantage™ NewSQL Engine Security Administration*, B035-1100
- For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.
- To disable the AVG extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on nulls, see *Teradata Vantage™ SQL Fundamentals*, B035-1141 and [Aggregates and Nulls](#).
- Aggregate user-defined functions (UDFs), see "Aggregate UDF" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210.
- Window aggregate UDFs, see "Window Aggregate UDF" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210.

## CORR

### Purpose

Returns the Sample Pearson product moment correlation coefficient of its arguments for all non-null data point pairs.

### Syntax

—— CORR —— ( *value\_expression\_1*, *value\_expression\_2* ) ——

## Syntax Elements

*value\_expression\_2/value\_expression\_1*

A numeric expression to be correlated with a second numeric expression.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The Sample Pearson product moment correlation coefficient is a measure of the linear association between variables. The boundary on the computed coefficient ranges from -1.00 to +1.00.

Note that high correlation does not imply a causal relationship between the variables.

The following table indicates the meaning of four extreme values for the coefficient of correlation between two variables.

IF the correlation coefficient has this value ...	THEN the association between the variables ...
-1.00	is perfectly linear, but inverse. As the value for y varies, the value for x varies identically in the opposite direction.
0	does not exist and they are said to be uncorrelated.
+1.00	is perfectly linear. As the value for y varies, the value for x varies identically in the same direction.
NULL	cannot be measured because there are no non-null data point pairs in the data used for the computation.

## Computation

The equation for computing CORR is defined as follows:

This variable ...	Represents ...
x	<i>value_expression_2</i>
y	<i>value_expression_1</i>

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for CORR(y, x) are as follows.

Data Type	Format	Title
REAL	the default format for DECIMAL(7,6)	CORR(y,x)

For an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including CORR, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE.

## Combination With Other Functions

CORR can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For information on ordered analytical functions, see [Ordered Analytical Functions](#).

CORR cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Example: Querying Data from the HomeSales Table

This example uses the data from the HomeSales table.

SalesPrice	NbrSold	Area
-----	-----	-----
160000	126	358711030

180000	103	358711030
200000	82	358711030
220000	75	358711030
240000	82	358711030
260000	40	358711030
280000	20	358711030

Consider the following query.

```
SELECT CAST (CORR(NbrSold,SalesPrice) AS DECIMAL (6,4))
FROM HomeSales
WHERE area = 358711030
AND SalesPrice Between 160000 AND 280000;

CORR(NbrSold,SalesPrice)
-----
-.9543
```

The result -.9543 suggests an inverse relationship between the variables. That is, for the area and sales price range specified in the query, the value for NbrSold increases as sales price decreases and decreases as sales price increases.

## CORR Window Function

For the CORR window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## Related Topics

For more information, see:

- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .
- For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

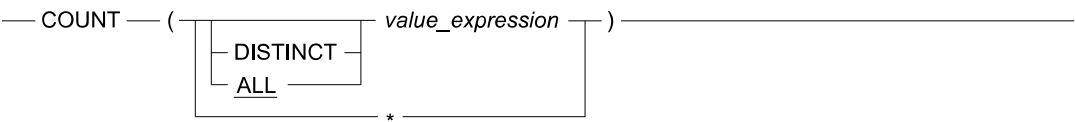
## COUNT

### Purpose

Returns a column value that is the total number of qualified rows in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

Syntax



Syntax Elements

ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

DISTINCT

Exclude duplicates specified by *value\_expression* from the computation.  
The expression cannot contain any ordered analytical or aggregate functions.

*value\_expression*

A literal or column expression for which the number of values is to be counted.  
The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

\*

Counts all rows in the group of rows on which COUNT operates.

Usage Notes

This syntax ...	Counts the total number of rows ...
COUNT( <i>value_expression</i> )	in the group for which <i>value_expression</i> is not null.
COUNT (DISTINCT <i>value_expression</i> )	in the group for which <i>value_expression</i> is unique and not null.
COUNT(*)	in the group of rows on which COUNT operates.

COUNT is valid for any data type.

## Result Type and Attributes

The following table lists the data type and format for the result of COUNT.

Mode	Data Type and Format
ANSI	MaxDecimal is general field 13 in the DBS Control utility. If MaxDecimal in DBSControl is... <ul style="list-style-type: none"> <li>• 0 or 15, then the result type is DECIMAL(15,0) and the format is -(15)9.</li> <li>• 18, then the result type is DECIMAL(18,0) and the format is -(18)9.</li> <li>• 38, then the result type is DECIMAL(38,0) and the format is -(38)9.</li> </ul>
Teradata	INTEGER and the format is the default format for INTEGER.
COUNT	The default value for the DBSControl General Field(80), COUNT_mode, is 0. The default is compatibility mode, which disables all extensions that impact external applications.

BIGINT and NUMBER modes impact COUNT performance:

- Type promotion may entail computing expressions using a different type if the mode is changed. This occurs when the result of the COUNT (\*) based expression is materialized as a BIGINT/NUMBER type, and later used as a subexpression for computing another expression. The performance overhead is the same as that incurred when casting COUNT (\*) as BIGINT/NUMBER.
- Since the data type of COUNT (\*) changes if the mode is changed, queries that made assumptions on format, title, and data type must be aware of the change.

If the result of COUNT overflows and reports an error, you can cast the result to another data type, as illustrated by the following example.

```
SELECT CAST(COUNT(*) AS BIGINT)
FROM BIGTABLE;
```

A similar example is provided for COUNT and rank window functions:

```
SELECT CAST(COUNT(*) over([PARTITION/ORDER BY]) AS BIGINT)
FROM BIGTABLE;
SELECT CAST(rank over([PARTITION/ORDER BY]) AS BIGINT)
FROM BIGTABLE;
```

### Note:

The CAST is required only for default or compatibility mode. If value of 1 or 2 is specified for NUMBER or BIGINT mode of computing COUNT, then the CAST is not required.

The following table lists the default title for the result of COUNT.

Operation	Title
COUNT(x)	Count(x)
COUNT(*)	Count(*)

### COUNT Specification in Aggregate Join Index

You can specify COUNT, COUNT cast to FLOAT OR DECIMAL(38,0), BIGINT, or NUMBER for a COUNT aggregate function in a join index. The following illustrates a SHOW JOIN INDEX that accommodates data type casts to BIGINT:

```
CREATE JOIN INDEX TEST.j1 ,NO FALLBACK ,CHECKSUM = DEFAULT AS
SELECT COUNT (*) (BIGINT, NAMED a ), TEST.t1.a1
FROM TEST.t1
GROUP BY TEST.t1.a1
PRIMARY INDEX ( a1 );
```

## Examples: Using the COUNT Function

### Example: Reporting the Number of Employees in Each Department

COUNT(\*) reports the number of employees in each department because the GROUP BY clause groups results by department number.

```
SELECT DeptNo, COUNT(*) FROM Employee
GROUP BY DeptNo
ORDER BY DeptNo;
```

Without the GROUP BY clause, only the total number of employees represented in the Employee table is reported:

```
SELECT COUNT(*) FROM Employee;
```

Note that without the GROUP BY clause, the select list cannot include the DeptNo column because it returns any number of values and COUNT(\*) returns only one value.

### Example: Employees Returned as Nulls

If any employees have been inserted but not yet assigned to a department, the return includes them as nulls in the DeptNo column.

```
SELECT DeptNo, COUNT(*) FROM Employee
GROUP BY DeptNo
ORDER BY DeptNo;
```

Assuming that two new employees are unassigned, the results table is:

DeptNo	Count(*)
-----	-----
?	2
100	4
300	3
500	7
600	4
700	3

### Example: Counting Employees Not Yet Assigned to a Department

If you ran the report in Example: Reporting the Number of Employees in Each Department using `SELECT... COUNT ...` without grouping the results by department number, the results table would have only registered non-null occurrences of `DeptNo` and would not have included the two employees not yet assigned to a department(nulls). The counts differ (23 in Example: Reporting the Number of Employees in Each Department as opposed to 21 using the statement documented in this example).

Recall that in addition to the 21 employees in the `Employee` table who are assigned to a department, there are two new employees who are not yet assigned to a department (the row for each new employee has a null department number).

```
SELECT COUNT(deptno) FROM employee ;
```

The result of this `SELECT` is that `COUNT` returns a total of the non-null occurrences of department number. Because aggregate functions ignore nulls, the two new employees are not reflected in the figure.

Count(DeptNo)
-----
21

### Example: Using COUNT to Find the Number of Employees by Gender

This example uses `COUNT` to provide the number of male employees in the `Employee` table of the database.

```
SELECT COUNT(sex)
FROM Employee
WHERE sex = 'M' ;
```

The result is as follows.

Count(Sex)
-----
12

**Example: Providing a Total of the Rows with Non-Null Department Numbers**

In this example COUNT provides, for each department, a total of the rows that have non-null department numbers.

```
SELECT deptno, COUNT(deptno)
FROM employee
GROUP BY deptno
ORDER BY deptno ;
```

Notice once again that the two new employees are not included in the count.

DeptNo	Count(DeptNo)
-----	-----
100	4
300	3
500	7
600	4
700	3

**Example: Returning the Number of Employees by Department**

To get the number of employees by department, use COUNT(\*) with GROUP BY and ORDER BY clauses.

```
SELECT deptno, COUNT(*)
FROM employee
GROUP BY deptno
ORDER BY deptno ;
```

In this case, the nulls are included, indicated by QUESTION MARK.

DeptNo	Count(*)
-----	-----
?	2
100	4
300	3
500	7
600	4
700	3

**Example: Determining the Number of Departments in the Employee Table**

To determine the number of departments in the Employee table, use COUNT (DISTINCT) as illustrated in the following SELECT COUNT.

```
SELECT COUNT (DISTINCT DeptNo)
FROM Employee ;
```

The system responds with the following report.

```
Count(Distinct(DeptNo))
-----
5
```

For time series examples, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## Related Topics

For more information, see:

- For COUNT functions that return the group, cumulative, or moving count, see [Window Aggregate Functions](#).
- With the exception of COUNT(\*), the computation does not include nulls. For more information, see *Teradata Vantage™ SQL Fundamentals*, B035-1141 and [Aggregates and Nulls](#).
- For information on data type default formats, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For information on the COUNT\_mode field, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## COVAR\_POP

### Purpose

Returns the population covariance of its arguments for all non-null data point pairs.

### Syntax

```
— COVAR_POP — ( value_expression_1, value_expression_2 ) —————
```

## Syntax Elements

*value\_expression\_1/value\_expression\_2*

A numeric expression to be paired with a second numeric expression to determine their covariance.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

Covariance measures whether or not two random variables vary in the same way. It is the average of the products of deviations for each non-null data point pair.

Note that high covariance does not imply a causal relationship between the variables.

## Combination With Other Functions

COVAR\_POP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause.

COVAR\_POP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are no non-null data point pairs in the data used for the computation, then COVAR\_POP returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for COVAR\_POP are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is a UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including COVAR\_POP, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .

## COVAR\_POP Window Function

For the COVAR\_POP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## COVAR\_SAMP

### Purpose

Returns the sample covariance of its arguments for all non-null data point pairs.

### Syntax

— COVAR\_SAMP — ( *value\_expression\_1*, *value\_expression\_2* ) —————

## Syntax Elements

*value\_expression\_2*/*value\_expression\_1*

A numeric expression to be paired with a second numeric expression to determine their covariance.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

Covariance measures whether or not two random variables vary in the same way. It is the sum of the products of deviations for each non-null data point pair.

Note that high covariance does not imply a causal relationship between the variables.

## Combination with Other Functions

COVAR\_SAMP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

COVAR\_SAMP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are no non-null data point pairs in the data used for the computation, then COVAR\_SAMP returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for COVAR\_SAMP(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is a UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including COVAR\_SAMP, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .

## COVAR\_SAMP Window Function

For the COVAR\_SAMP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Using the SELECT statement to Return the Sample Covariance of Weight and Height

This example is based on the following regtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the sample covariance of weight and height where neither weight nor height is null.

```
SELECT COVAR_SAMP(weight,height)
FROM regtbl;
```

```
Covar_Samp(weight,height)
-----
150
```

# GROUPING

## Purpose

Returns a value that indicates whether a specified column in the result row was excluded from the grouping set of a GROUP BY clause.

## Syntax

```
—— GROUPING —— ( expression ) ——
```

## Syntax Elements

*expression*

A column in the result row that might have been excluded from a grouped query containing CUBE, ROLLUP, or GROUPING SET.

The argument must be an item of a GROUP BY clause.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Usage Notes

A null in the result row of a grouped query containing CUBE, ROLLUP, or GROUPING SET can mean one of the following:

- The actual data for the column is null.
- The extended grouping specification aggregated over the column and excluded it from the particular grouping. A null in this case really represents all values for this column.

Use GROUPING to distinguish between rows with nulls in actual data from rows with nulls generated from grouping sets.

## Result Type and Attributes

The data type, format, and title for GROUPING(x) are as follows.

Data Type	Format	Title
INTEGER	Default format of the INTEGER data type	Grouping(x)

## Result Value

IF the value of the specified column in the result row is ...	THEN GROUPING returns ...
a NULL generated when the extended grouping specification aggregated over the column and excluded it from the particular grouping	1
anything else	0

## Example: Viewing Sales Summaries by County and by City

Suppose you have the following data in the sales\_view table.

PID	Cost	Sale	Margin	State	County	City
1	38350	50150	11800	CA	Los Angeles	Long Beach
1	63375	82875	19500	CA	San Diego	San Diego
1	46800	61200	14400	CA	Los Angeles	Avalon
2	40625	53125	12500	CA	Los Angeles	Long Beach

To look at sales summaries by county and by city, use the following SELECT statement:

```
SELECT county, city, sum(margin)
FROM sale_view
GROUP BY GROUPING SETS ((county),(city));
```

The query reports the following data:

County	City	Sum(margin)
-----	-----	-----
Los Angeles	?	38700
San Diego	?	19500
?	Long Beach	24300
?	San Diego	19500
?	Avalon	14400

Notice that in this example, a null represents all values for a column because the column was excluded from the grouping set represented.

To distinguish between rows with nulls in actual data from rows with nulls generated from grouping sets, use the GROUPING function:

```
SELECT county, city, sum(margin),
       GROUPING(county) AS County_Grouping,
```

```

      GROUPING(city) AS City_Grouping
FROM sale_view
GROUP BY GROUPING SETS ((county),(city));

```

The results are:

County	City	Sum(margin)	County_Grouping	City_Grouping
-----	-----	-----	-----	-----
Los Angeles	?	38700	0	1
San Diego	?	19500	0	1
?	Long Beach	24300	1	0
?	San Diego	19500	1	0
?	Avalon	14400	1	0

You can also use GROUPING to replace the nulls that appear in a result row because the extended grouping specification aggregated over a column and excluded it from the particular grouping. For example:

```

SELECT CASE
      WHEN GROUPING(county) = 1
      THEN '-All Counties-'
      ELSE county
    END AS County,
    CASE
      WHEN GROUPING(city) = 1
      THEN '-All Cities-'
      ELSE city
    END AS City,
    SUM(margin)
FROM sale_view
GROUP BY GROUPING SETS (county,city);

```

The query reports the following data:

County	City	Sum(margin)
-----	-----	-----
Los Angeles	-All Cities-	38700
San Diego	-All Cities-	19500
-All Counties-	Long Beach	24300
-All Counties-	San Diego	19500
-All Counties-	Avalon	14400

## Related Topics

For more information on GROUP BY, GROUPING SETS, ROLLUP, and CUBE, see *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## KURTOSIS

### Purpose

Returns the kurtosis of the distribution of *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

— KURTOSIS — ( ( DISTINCT | ALL ) value_expression )

```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

Exclude duplicates specified by *value\_expression* from the computation.

### *value\_expression*

A literal or column expression for which the kurtosis of the distribution of its values is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Definition

Kurtosis is the fourth moment of the distribution of the standardized (z) values. It is a measure of the outlier (rare, extreme observation) character of the distribution as compared with the normal, Gaussian distribution.

The normal distribution has a kurtosis of 0.

Positive kurtosis indicates that the distribution is more outlier-prone than the normal distribution, while negative kurtosis indicates that the distribution is less outlier-prone than the normal distribution.

## Return Value

This function returns the REAL data type.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on a UDT argument that has an implicit cast that casts between the UDT and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including KURTOSIS, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .

## Computation

The equation for computing KURTOSIS is defined as follows:

$$\text{Kurtosis} = \left( \frac{(\text{COUNT}(x))(\text{COUNT}(x) + 1)}{(\text{COUNT}(x) - 1)(\text{COUNT}(x) - 2)(\text{COUNT}(x) - 3)} \right) \left( \text{SUM} \left( \frac{x - \text{AVG}(x)}{\text{STDEV\_SAMP}(x)}^{**4} \right) \right) - \left( \frac{(3)((\text{COUNT}(x) - 1)(**2))}{(\text{COUNT}(x) - 2)(\text{COUNT}(x) - 3)} \right)$$

where:

This variable ...	Represents ...
x	<i>value_expression</i>

## Conditions That Produce a NULL Return Value

The following conditions produce a null return value:

- Fewer than four non-null data points in the data used for the computation
- $\text{STDDEV\_SAMP}(x) = 0$
- Division by zero

## MAX

### Purpose

Returns a column value that is the maximum value for *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

— MAXIMUM — ( — DISTINCT — value_expression )
— MAX —      — ALL —

```

## Syntax Elements

### ALL

All values that are not null specified by *value\_expression*, including duplicates, are included in the maximum value computation for the group. This is the default.

### DISTINCT

Exclude duplicates specified by *value\_expression* from the computation.

Duplicate and values that are not null specified by *value\_expression* are eliminated from the maximum value computation for the group.

*value\_expression*

A literal or column expression for which the maximum value is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

MAXIMUM is a Teradata extension to the ANSI SQL:2011 standard.

## Result Type and Attributes

The following table lists the default attributes for the result of MAX(x).

Attribute	Value
Data Type	If operand x is not a UDT, the result data type is the data type of operand x. If operand x is a UDT, the result data type is the data type to which the UDT is implicitly cast.
Format	If operand x is not a UDT, the result data type is the data type of operand x. If operand x is a UDT, the result data type is the data type to which the UDT is implicitly cast.
Title	Maximum(x)

## Support for UDTs

By default, Teradata Database performs implicit type conversion on a UDT argument that has an implicit cast that casts between the UDT and any of the following predefined types:

- Numeric
- Character
- Byte
- DATE
- TIME or TIMESTAMP
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including MAX, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the

DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .

## Usage Notes

MAX is valid for character data as well as numeric data. When used with a character expression, MAX returns the highest sort order.

Nulls are not included in the result computation. For more information, see *Teradata Vantage™ SQL Fundamentals*, B035-1141 and [Aggregates and Nulls](#).

If *value\_expression* is a column expression, the column must refer to at least one column in the table from which data is selected.

The *value\_expression* must not specify a column reference to a view column that is derived from a function.

## MAX Window Function

For the MAX window function that computes a group, cumulative, or moving maximum value, see [Window Aggregate Functions](#).

## Examples: Using the MAX Function

### Example: CHARACTER Data

The following SELECT returns the immediately following result.

```
SELECT MAX(Name)
FROM Employee;

Maximum(Name)
-----
Zorn J
```

### Example: Column Expressions

You want to know which item in your warehouse stock has the maximum cost of sales.

```
SELECT MAX(CostOfSales) AS m, ProdID
FROM Inventory
GROUP BY ProdID
ORDER BY m DESC;

Maximum(CostOfSales)  ProdID
-----
```

1295	3815
975	4400
950	4120

For time series examples, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## MIN

### Purpose

Returns a column value that is the minimum value for *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

— MINIMUM ( [ DISTINCT | ALL ] value_expression )
— MIN

```

## Syntax Elements

### ALL

All values that are not null specified by *value\_expression*, including duplicates, are included in the minimum value computation for the group. This is the default.

### DISTINCT

Exclude duplicates specified by *value\_expression* from the computation.

Duplicate and values that are not null specified by *value\_expression* are eliminated from the minimum value computation for the group.

### *value\_expression*

A literal or column expression for which the minimum value is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

MINIMUM is a Teradata extension to the ANSI SQL:2011 standard.

## Result Type and Attributes

The following table lists the default attributes for the result of MIN(x).

Attribute	Value
Data type	If operand x is not a UDT, the result data type is the data type of operand x. If operand x is a UDT, the result data type is the data type to which the UDT is implicitly cast.
Title	Minimum(x)
Format	If operand x is not a UDT, the result format is the format of operand x. If operand x is a UDT, the result format is the format of the data type to which the UDT is implicitly cast.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on a UDT argument that has an implicit cast that casts between the UDT and any of the following predefined types:

- Numeric
- Character
- Byte
- DATE
- TIME or TIMESTAMP
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including MIN, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143 .

## Usage Notes

MINIMUM is valid for character data as well as numeric data. MINIMUM returns the lowest sort order of a character expression.

The computation does not include nulls. For more information, see “Manipulating Nulls” in *Teradata Vantage™ SQL Fundamentals*, B035-1141 and [Aggregates and Nulls](#).

If *value\_expression* specifies a column expression, the expression must refer to at least one column in the table from which data is selected.

If *value\_expression* specifies a column reference, the column must not be a view column that is derived from a function.

## MIN Window Function

For the MIN window function that computes a group, cumulative, or moving minimum value, see [Window Aggregate Functions](#).

## Examples: Using the MINIMUM Function

### Example: MINIMUM Used With CHARACTER Data

The following SELECT returns the immediately following result.

```
SELECT MINIMUM(Name)
FROM Employee;

Minimum(Name)
-----
Aarons A
```

### Example: JIT Inventory

Your manufacturing shop has recently changed vendors and you know that you have no quantity of parts from that vendor that exceeds 20 items for the ProdID. You need to know how many of your other inventory items are low enough that you need to schedule a new shipment, where “low enough” is defined as fewer than 30 items in the QUANTITY column for the part.

```
SELECT ProdID, MINIMUM(QUANTITY)
FROM Inventory
WHERE QUANTITY BETWEEN 20 AND 30
GROUP BY ProdID
ORDER BY ProdID;
```

The report is as follows:

ProdID	Minimum(Quantity)
1124	24
1355	21
3215	25
4391	22

For time series examples, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

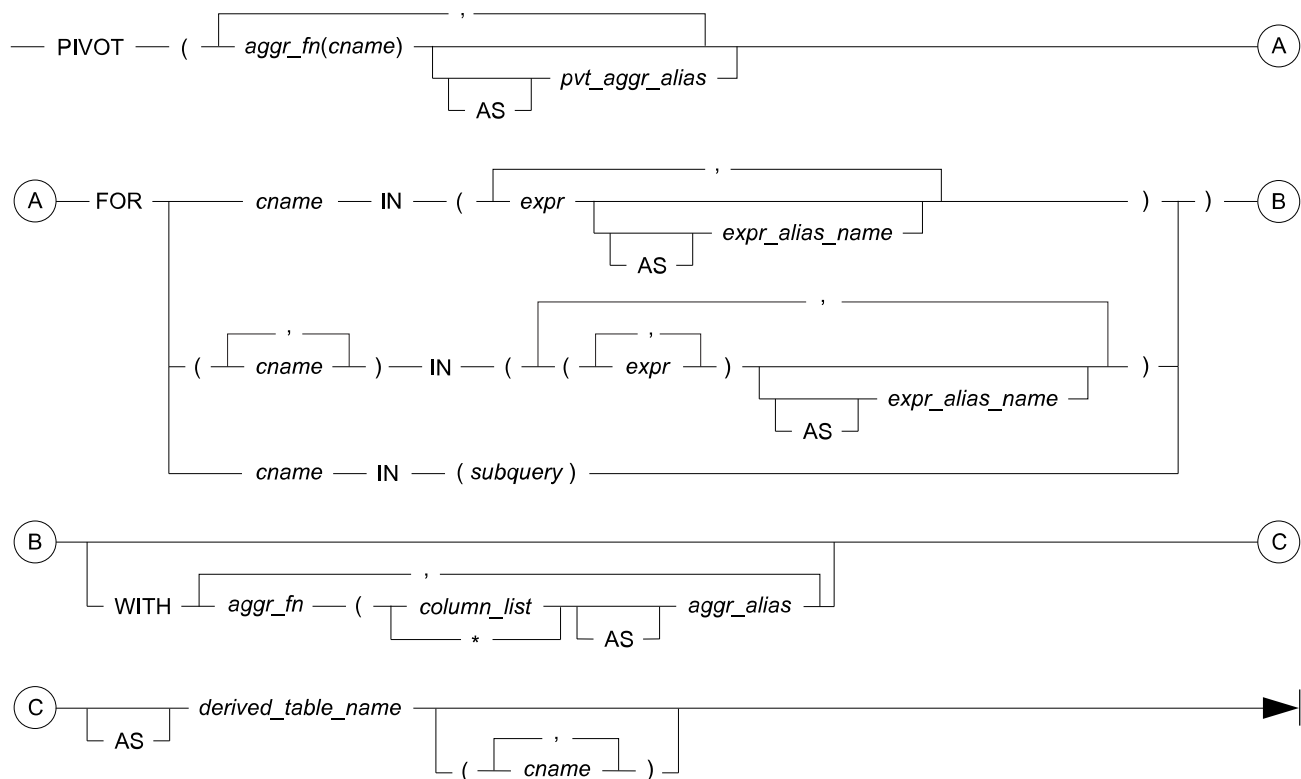
# PIVOT

## Purpose

PIVOT is a relational operator for transforming rows into columns. The function is useful for reporting purposes, as it allows you to aggregate and rotate data to create easy-to-read tables. You can perform PIVOT aggregation on PIVOT column results by using the WITH clause.

Specify the PIVOT operator in the FROM clause of the SELECT statement. There are no restrictions on other clauses that can be specified with the SELECT query that include PIVOT operators.

## Syntax



## Syntax Elements

*aggr\_fn*

An aggregate function that supports a single argument.

*pvt\_aggr\_alias*

An alias name specified for the Aggregate function.

*expr\_alias\_name*

An alias name specified for the values/expressions specified in the IN list.

*cname*

A column name.

*derived\_table\_name*

The table name specified for the resultant pivoted table.

*expr*

An expression or a column value.

## WITH

Using the WITH clause, you can specify all Pivot columns using an asterisk (\*) or a subset of columns on which the aggregation function needs to perform.

*aggr\_fn*

An aggregate function.

*column\_list*

A list of one or more columns. If the list contains more than one column, separate them with commas.

\*

Option to include all the Pivot columns without specifying columns explicitly.

**alias**

Name of the aggregate result column.

## Usage Notes

---

**Note:**

For the PIVOT operation, column names within the Aggregate functions are referred to as measure columns, and column names in the FOR clause are referred to as pivot columns.

---

As indicated in the syntax, specify at least one Aggregate function with the PIVOT operator.

Columns with CLOB, BLOB, UDT, XML, or JSON data types are not allowed with the PIVOT operator.

Column names are not allowed within the IN-list. Only values or expressions (arithmetic expressions such as MOD or ABS, or string Manipulation expressions such as LENGTH, REVERSE) are allowed.

Measure columns and pivot columns of the PIVOT operator are not allowed in the assign list of the SELECT statement.

If  $n$  number of Aggregate functions are specified where  $n$  is greater than 1, then the alias name must be specified for at least  $(n-1)$  aggregate functions.

The *cname* specified in the *derived\_table\_name* takes precedence over the alias names derived from the IN-list.

If the alias names are not specified for the column values listed in the IN clause, the database processing encloses the column values into double quotes and converts these string literals to alias names using the default format. The alias names are used as column names of the pivoted table.

If the length of the alias name derived from a column value exceeds the alias name limit of 128 characters (if EON feature is enabled) or 30 characters (if EON is not enabled), the alias name is truncated.

If the IN-list contains case-specific values such as 'abc' & 'ABC', the values are treated the same and an error occurs.

PIVOT supports the UNPIVOT or TD\_UNPIVOT functions as a query source for the PIVOT operator.

The PIVOT/UNPIVOT operator uses a single dimensional way of converting rows to columns, or columns to rows. You can swap both rows and columns within a single query (for example, using UNPIVOT as source to PIVOT). This provides flexibility when using the two-dimensional method of interchanging data in a table.

Using the DT column list for UNPIVOT as a query source to PIVOT is optional.

If the WITH clause is specified in the PIVOT query:

- Specifying at least one aggregate function with the WITH operator is mandatory.
- SUM, AVG, MIN, and MAX aggregate functions are supported.
- The *cname* specified in the *derived\_table\_name* takes precedence over the alias names derived for the aggregated result columns.
- DISTINCT keyword is not supported with aggregate column.

- Column list is not allowed if an asterisk (\*) is specified.
- Aggregating a column list or \* may produce meaningless results if the values aggregated are not related. For example, if some pivot columns are for SUM and some are for an AVG, WITH SUM(\*) is not a meaningful value.
- Column names mentioned in the aggregate function should be PIVOT columns or subset of PIVOT columns.

To avoid the overhead of issuing a separate query to generate values for input to the PIVOT IN-list clause as hard coded constants, you can issue the query as a subquery in the PIVOT IN-list. If a PIVOT query has a subquery in the IN-list:

- Alias names are not allowed in the IN-list.
- Alias names in the PIVOT derived table are not allowed.
- The SELECT list of the subquery must contain only one column reference.
- The subquery must return at least one row.
- The results returned by the subquery cannot exceed 32KB, and the row count must be less than or equal to 16.
- SET operations are not allowed on a PIVOT query that has a subquery in the IN-list.
- Columns generated by an IN-list subquery cannot be explicitly used in the SELECT.
- You cannot use a subquery in a PIVOT IN-list with DDL statements or multistatement requests.
- A PIVOT query cannot include both a WITH clause and a subquery in the IN-list.

For examples of wide tables, see [Pivot Examples](#).

## Examples

### Example: Alias Names Contained in the IN List

This example uses the *star1* table, with the following definition and contents:

```
CREATE TABLE star1(country VARCHAR(20),state VARCHAR(10), yr INTEGER,qtr
VARCHAR(3),sales INTEGER,cogs INTEGER);
```

```
SELECT * FROM star1;
```

country	state	yr	qtr	sales	cogs
-----	-----	-----	---	-----	-----
USA	CA	2001	Q1	30	15
Canada	ON	2001	Q2	10	0
Canada	BC	2001	Q3	10	0
USA	NY	2001	Q1	45	25
USA	CA	2001	Q2	50	20

In this example, the IN list contains alias names. The alias names are concatenated with the alias names specified by the aggregate functions to build the column names of the output pivoted table.

```

SELECT *
FROM star1 PIVOT (
                SUM(sales) as ss1, SUM(cogs) as sc FOR
qtr
                IN ('Q1' AS
Quarter1,
                'Q2' AS Quarter2,
                'Q3' AS Quarter3)
                )Tmp;

```

The output is re-written as an equivalent SELECT query using CASE statements:

```

SELECT * FROM (SELECT country ,state ,yr ,
SUM(CASE WHEN qtr = 'Q1' THEN sales ELSE NULL END )AS Quarter1_ss1,
SUM(CASE WHEN qtr = 'Q1' THEN (cogs) ELSE NULL END )AS Quarter1_sc,
SUM(CASE WHEN qtr = 'Q2' THEN (sales) ELSE NULL END)AS Quarter2_ss1,
SUM(CASE WHEN qtr = 'Q2' THEN (cogs) ELSE NULL END)AS Quarter2_sc,
SUM(CASE WHEN qtr = 'Q3' THEN (sales) ELSE NULL END)AS Quarter3_ss1,
SUM(CASE WHEN qtr = 'Q3' THEN (cogs) ELSE NULL END)AS Quarter3_sc
FROM star1 GROUP BY country ,state ,yr ) Tmp ;

```

Output pivoted table:

country	state	yr	Quarter1_ss1	Quarter1_sc	Quarter2_ss1	Quarter2_sc	Quarter3_ss1	Quarter3_sc
-----	-----	-----	-----	-----	-----	-----	-----	-----
-----	-----	-----	-----	-----	-----	-----	-----	-----
USA	CA	2001	30	15	50	20	?	?
USA	NY	2001	45	25	?	?	?	?
Canada	ON	2001	?	?	10	0	?	?
Canada	BC	2001	?	?	?	?	?	?
10		0						

## Example: Naming Columns with the <column\_value\_list> Values

In this example, the SELECT statement does not specify the names to use for columns explicitly. The names of the columns are built internally by adding the aggregated column name to the <column\_value\_list> values.

```

SELECT *
FROM star1 PIVOT (SUM(sales) AS ss1, SUM(cogs) AS sc FOR (yr, qtr)
                IN ((2001, 'Q1'),
                (2001, 'Q2'),

```

```
(2001, 'Q3'))
)Tmp;
```

This is re-written as an equivalent SELECT query that uses CASE statements:

```
SELECT * FROM (SELECT country ,state ,
SUM(CASE WHEN yr = 2001 AND qtr = 'Q1' THEN sales ELSE NULL END) AS
"2001_'Q1'_'ss1" ,
SUM(CASE WHEN yr = 2001 AND qtr = 'Q1' THEN cogs ELSE NULL END) AS
"2001_'Q1'_'sc",
SUM(CASE WHEN yr = 2001 AND qtr = 'Q2' THEN sales ELSE NULL END) AS
"2001_'Q2'_'ss1" ,
SUM(CASE WHEN yr = 2001 AND qtr = 'Q2' THEN cogs ELSE NULL END) AS
"2001_'Q2'_'sc",
SUM(CASE WHEN yr = 2001 AND qtr = 'Q3' THEN sales ELSE NULL END) AS
"2001_'Q3'_'ss1",
SUM(CASE WHEN yr = 2001 AND qtr = 'Q3' THEN cogs ELSE NULL END) AS "2001_'Q3'_'sc"
FROM star1 GROUP BY country ,state ) Tmp ;
```

Output pivoted table:

```
country state 2001_'Q1'_'ss1 2001_'Q1'_'sc 2001_'Q2'_'ss1 2001_'Q2'_'sc
2001_'Q3'_'ss1 2001_'Q3'_'sc
-----
-----
USA      CA          30          15          50          20          ?          ?
USA      NY          45          25           ?           ?           ?           ?
Canada   ON           ?           ?          10           0           ?           ?
Canada   BC           ?           ?           ?           ?           ?
10        0
```

## Example: Pivot Operation on View

The following example of a view as a PIVOT source.

Assume a view, *v1*, is defined on the table *s1*:

```
CREATE TABLE s1(yr INTEGER, mon VARCHAR(4), sales INTEGER);

sel * from s1;

sel * from s1;

*** Query completed. 8 rows found. 3 columns returned.
*** Total elapsed time was 1 second.
```

yr	mon	sales
2001	jan	100
2003	jan	300
2002	jan	150
2001	feb	110
2003	feb	310
2002	feb	200
2001	mar	120
2002	mar	250

```
CREATE VIEW V1 AS select yr,sales from s1;
```

```
*** View has been created.
```

```
*** Total elapsed time was 1 second.
```

```
sel * from v1;
```

```
select * from v1;
```

```
*** Query completed. 8 rows found. 2 columns returned.
```

```
*** Total elapsed time was 1 second.
```

yr	sales
2002	150
2003	300
2002	200
2003	310
2002	250
2001	100
2001	110
2001	120

The following query generates sales report with respect to each year on view V1:

```
SELECT *
FROM v1 PIVOT (SUM(sales) FOR yr IN (2001,2002,2003)) tmp;
```

```
*** Query completed. One row found. 3 columns returned.
```

```
*** Total elapsed time was 1 second.
```

2001	2002	2003
-----	-----	-----
330	600	610

### Example: Table Source Using the WITH Clause

The following is an example of a table using the WITH clause as a source to the pivot query.

```
SELECT *
FROM (with temp
as (select * from s1) select * from temp)dt PIVOT (SUM(sales) FOR mon IN
('Jan','Feb', 'Mar'))tmp;
```

\*\*\* Query completed. 3 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

yr	Jan	Feb	Mar
-----	-----	-----	-----
2001	100	110	120
2002	150	200	250
2003	300	310	?

### Example: SELECT Query with the WHERE Condition

The following is an example of using a SELECT query with the WHERE condition:

```
SELECT *
FROM s1 PIVOT (SUM(sales) FOR mon IN ('Jan' as Jan, 'Feb' as Feb, 'Mar' as
Mar))tmp where Jan=100;
```

\*\*\* Query completed. 1 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

Yr	Jan	Feb	Mar
-----	-----	-----	-----
2001	100	110	120

### Example: CREATE TABLE AS Statement Contains Special Characters

In this example, the CREATE TABLE AS statement contains special characters in the pivot query IN list.

```
CREATE TABLE t1 AS
(SELECT *
FROM s1 PIVOT (SUM(sales) FOR mon IN (U&"#FAD7" UESCAPE '#')) tmp ) WITH DATA;
```

```
*** Failure 4306 Invalid PIVOT query: Unsupported In-List Values/Expressions.
      Statement# 1, Info =0
*** Total elapsed time was 1 second.
```

## Example: The PIVOT Query Response in Different Response Modes

Assume a table t1 is defined as:

```
CREATE TABLE t1(yr INTEGER,mon VARCHAR(3),sales INTEGER);
```

Assume that following insert statements

```
INSERT t1 VALUES(2003,'Jan',300);
INSERT t1 VALUES(2001,'Jan',100);
INSERT t1 VALUES(2003,'Feb',310);
INSERT t1 VALUES(2001,'Feb',110);
INSERT t1 VALUES(2002,'Jan',150);
INSERT t1 VALUES(2001,'Mar',120);
INSERT t1 VALUES(2002,'Feb',200);
INSERT t1 VALUES(2002,'Mar',250);
INSERT t1 VALUES(2003,'Mar',1000);
```

Assuming that the PIVOT query is submitted for execution, the output returns as different responses modes.

For a PIVOT query:

```
SELECT * FROM t1 PIVOT(SUM(sales) FOR mon IN ('Jan','Feb','Mar')) tmp;
```

For a PIVOT query re-written as a SELECT statement using CASE expressions:

```
SELECT yr,SUM(case when mon='Jan' then sales end) AS "Jan",
SUM(case when mon='Feb' then sales end) AS "Feb",
SUM(case when mon='Mar' then sales end) AS "Mar"
FROM t1 GROUP BY yr;
```

.field mode

```
*** Query completed. 3 rows found.
```

yr	Jan	Feb	Mar
2001	100	110	120
2003	300	310	1000
2002	150	200	250

```
.multipartrecord mode
```

```
*** Query completed. 3 rows found.
```

yr	Jan	Feb	Mar
2001	100	110	120
2003	300	310	1000
2002	150	200	250

```
.record mode
```

```
*** Query completed. 3 rows found.
```

yr	Jan	Feb	Mar
2001	100	110	120
2003	300	310	1000
2002	150	200	250

```
.indicator mode
```

```
*** Query completed. 3 rows found.
```

yr	Jan	Feb	Mar
2001	100	110	120
2003	300	310	1000
2002	150	200	250

## Example: Pivot Query Truncates the Alias Name

For the first part of this example, the EnableEON dbscontrol flag is set to false, so the column name limit defaults to 30 characters.

Assume the table *t1* is defined as:

```
CREATE TABLE t1(yr INTEGER, mon VARCHAR(41), sales INTEGER);
```

Also assume that the table *t1* contains the following row:

```
SELECT * FROM t1;
yr      mon      sales
```

```

-----
2001      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa      200

```

The row contains 35 characters for the column 'mon'.

The following pivot query results truncate the 'mon' column value from 35 characters to 30 characters:

```

SELECT * FROM t1 PIVOT(SUM(sales) FOR mon IN
('aaaaaaaaaaaaaaaaaaaaaaaaaaaaa'))tmp;

```

```

YR      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-----
2001    200

```

Now, assume that the EnableEON dbscontrol flag is set to true, so the column name limit defaults to 128 characters.

Also assume that table *t2* is defined as follows:

```

CREATE TABLE t2(yr INTEGER, mon VARCHAR(131), sales INTEGER);

```

Assume that the table *t2* contains the following row:

```

SELECT mon FROM t2;
mon
-----
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```

The row contains 130 characters for the column 'mon'.

The following pivot query truncates the 'mon' column value from 130 characters to 128 characters:

```

SELECT * FROM t2 PIVOT(SUM(sales) FOR mon IN
('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')) tmp;

```

```

YR
---
2001
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-----
200

```

## Example: Using TD\_UNPIVOT or UNPIVOT as a Source to PIVOT

PIVOT supports UNPIVOT query or the TD\_UNPIVOT function as a source for the PIVOT operator.

PIVOT/ UNPIVOT uses a single dimensional method to interchange data, such as converting rows to columns, or columns to rows, based on some aggregation on a column data.

Swap rows and columns within a single query by giving UNPIVOT query as a source to PIVOT. This provides flexibility for a two-dimensional way of interchanging data in a table based on some aggregation on a column.

---

### Note:

To change data with a two-dimensional method, aggregate data on a column, and then interchange the rows and columns twice. In this case, swap rows and columns based on some aggregation on a column data. The table rotates twice by some aggregation, but might not return the actual table rows. It could introduce new rows where data is missing, or eliminate rows if data is aggregated in the process.

Two-dimensional uses PIVOT as source to the UNPIVOT query, or UNPIVOT as a source to a PIVOT query. Using PIVOT as source to an UNPIVOT query is complex when writing the SQL, whereas using UNPIVOT as a source to PIVOT query is easier.

---

First, create a table with the following data:

```
CREATE TABLE t1 (place CHAR(5), sales1 INTEGER, sales2 INTEGER,
                  sales3 INTEGER, sales4 INTEGER, sales5 INTEGER)
PRIMARY INDEX ( place );
```

place	sales1	sales2	sales3	sales4	sales5
Hyd	110	100	1000	1100	500
Che	120	200	2000	1200	600
Kol	150	500	5000	1500	900
Mee	140	400	4000	1400	800
Pun	130	300	3000	1300	700

To get the SUM of sales for each place, swap the sales and place using the following query:

```
SELECT * from (SELECT * from t1
                UNPIVOT(saleval
                        for sales in (sales1, sales2, sales3,
                                      sales4, sales5))dt1)dt2
                PIVOT(SUM(saleval)
                      for place in ('hyd','Che','pun',
                                    'mee','kol'))dt3;
```

The results for using UNPIVOT as the source:

sales	Hyd	Che	Pun	Mee	Kol
-----	-----	-----	-----	-----	-----
sales1	110	120	130	140	150
sales2	100	200	300	400	500
sales3	1000	2000	3000	4000	5000
sales4	1100	1200	1300	1400	1500
sales5	500	600	700	800	900

## Example: Aggregation on Two Columns from PIVOT Results

This example shows how to sum sales in the months of Jan and Feb for each year. This is an aggregation on two columns from the PIVOT result.

Table s1 is defined as:

```
CREATE TABLE s1 (yr INTEGER, mon VARCHAR(20), sales INTEGER);
```

The table contains:

```
SELECT * FROM s1;
yr      mon      sales
-----  ---      -
2001    Jan      100
2003    Jan      300
2002    Jan      150
2001    Feb      110
2003    Feb      310
2002    Feb      200
2001    Mar      120
2002    Mar      250
```

The PIVOT query is:

```
SELECT * FROM s1 PIVOT(SUM(SALES) FOR MON IN ('JAN', 'FEB', 'MAR')
  WITH SUM('JAN', 'FEB') AS AGGR1 ) DT
order by 1;
```

AGGR1 is the name of the aggregated result column.

Output:

yr	'JAN'	'FEB'	'MAR'	AGGR1
-----	-----	-----	-----	-----
2001	100	110	120	210
2002	150	200	250	350
2003	300	310	?	610

## Example: Subquery in PIVOT IN-List

This is an example of having a subquery in PIVOT IN-list.

Table s1 is defined as:

```
CREATE TABLE s1(yr INTEGER, mon VARCHAR (5), sales INTEGER);
CREATE TABLE s2(yr INTEGER, mon VARCHAR (5), sales INTEGER);
```

The table contains:

```
SELECT * FROM s1;
yr      mon      sales
-----  ---      -
2001    Jan      100
2003    Jan      300
2002    Jan      150
2001    Feb      110
2003    Feb      310
2002    Feb      200
2001    Mar      120
2002    Mar      250
```

```
SELECT * FROM s2;
yr      mon      sales
-----  ---      -
2001    Jan      100
2002    Mar      250
2003    Feb      310
```

The table as a source to a PIVOT query having a subquery in PIVOT IN-list:

```
SELECT * FROM s1 PIVOT (SUM (sales) FOR mon in (SELECT mon FROM s2)) dt;
```

The output pivoted table:

```
*** Query completed. 3 rows found. 4 columns returned.
*** Total elapsed time was 1 second.
```

yr	'Feb'	'Jan'	'Mar'
2001	110	100	120
2003	310	300	?
2002	200	150	250

## Related Topics

For more information, see [UNPIVOT](#).

## REGR\_AVGX

### Purpose

Returns the mean of the *independent\_variable\_expression* for all non-null data pairs of the dependent and independent variable arguments.

### Syntax

— REGR\_AVGX — ( *dependent\_variable\_expression*, *independent\_variable\_expression* ) —————

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_AVGX can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_AVGX cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_AVGX returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_AVGX(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is a UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_AVGX, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

## REGR\_AVGX Window Function

For the REGR\_AVGX window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Mean Height for regrtbl

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
--	-----	-----
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the mean height for regrtbl where neither weight nor height is null.

```
SELECT REGR_AVGX(weight,height)
FROM regrtbl;
```

```
Regr_Avgx(weight,height)
-----
                        68
```

## REGR\_AVGY

### Purpose

Returns the mean of the *dependent\_variable\_expression* for all non-null data pairs of the dependent and independent variable arguments.

**Syntax**

— REGR\_AVGY — ( *dependent\_variable\_expression*, *independent\_variable\_expression* ) —————

**Syntax Elements***dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

*independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

**ANSI Compliance**

This statement is ANSI SQL:2011 compliant.

**Setting Up Axes for Plotting**

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

**Combination With Other Functions**

REGR\_AVGY can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_AVGY cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

**Computation**

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_AVGY returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_AVGY(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is a UDT, the format is the format for the data type to which the UDT is implicitly cast.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_AVGY, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_AVGY Window Function

For the REGR\_AVGY window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Mean Weight from regrtbl

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
--	-----	-----
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119

6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the mean weight from regtbl where neither height nor weight is null.

```
SELECT REGR_AVGY(weight,height)
FROM regtbl;
```

```
Regr_Avgy(weight,height)
-----
                        140
```

## Related Topics

For more information, see *Teradata Vantage™ Data Types and Literals*, B035-1143:

- Information on the default format of data types and an explanation of the formatting characters in the format
- Information on implicit type conversion of UDTs

## REGR\_COUNT

### Purpose

Returns the count of all non-null data pairs of the dependent and independent variable arguments.

### Syntax

```
— REGR_COUNT — ( dependent_variable_expression, independent_variable_expression ) —————
```

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

*independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_COUNT can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_COUNT cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Result Type and Attributes

The following table lists the data type for the result of REGR\_COUNT(y,x).

Mode	Data Type
ANSI	If MaxDecimal in DBSControl is... <ul style="list-style-type: none"> <li>• 0 or 15, then the result type is DECIMAL(15,0).</li> <li>• 18, then the result type is DECIMAL(18,0).</li> <li>• 38, then the result type is DECIMAL(38,0).</li> </ul>
Teradata	INTEGER

The result type of REGR\_COUNT is consistent with the result type of COUNT for ANSI transaction mode and Teradata transaction mode.

When in Teradata mode, if the result of REGR\_COUNT overflows and reports an error, you can cast the result to another data type, as illustrated by the following example.

```
SELECT CAST(REGR_COUNT(weight,height) AS BIGINT)
FROM regrtbl;
```

Here are default formats and titles for the result of REGR\_COUNT.

- If operand y is numeric or character, the format is:
  - For ANSI mode, if MaxDecimal in DBSControl is:  
 0 or 15, the format is -(15)9  
 18, the format is -(18)9  
 38, the format is -(38)9
  - For Teradata mode, the format is the default format for INTEGER
- If operand y is UDT, the format is the format for the data type to which the UDT is implicitly cast.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_COUNT, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_COUNT Window Function

For the REGR\_COUNT window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Number of Rows in regrtbl

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
1	60	84
2	62	95
3	64	140

c1	height	weight
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the number of rows in regrtbl where neither height nor weight is null.

```
SELECT REG_COUNT(weight,height)
FROM regrtbl;
```

Here is the result:

```
Regr_Count(weight,height)
-----
                        9
```

## Related Topics

- For information on data type default formats, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For information on the REGR\_COUNT window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## REGR\_INTERCEPT

### Purpose

Returns the intercept of the univariate linear regression line through all non-null data pairs of the dependent and independent variable arguments.

## Syntax

— REGR\_INTERCEPT — ( *dependent\_variable\_expression*, *independent\_variable\_expression* ) —

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The intercept is the point at which the regression line through the non-null data pairs in the sample intersects the ordinate, or y-axis, of the graph.

The plot of the linear regression on the variables is used to predict the behavior of the dependent variable from the change in the independent variable.

Note that this computation assumes a linear relationship between the variables.

There can be a strong nonlinear relationship between independent and dependent variables, and the computation of the simple linear regression between such variable pairs does not reflect such a relationship.

## Independent and Dependent Variables

An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

A dependent variable is something that is measured in response to a treatment.

For example, you might want to test the ability of various promotions to enhance sales of a particular item.

In this case, the promotion is the independent variable and the sales of the item made as a result of the individual promotion is the dependent variable.

The value of the linear regression intercept tells you the predicted value for sales when there is no promotion for the item selected for analysis.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_INTERCEPT can be combined with any of the ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_INTERCEPT cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_INTERCEPT returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_INTERCEPT(y, x) are as follows.

Data Type	Format	Title
REAL	Default format of the REAL data type	REGR_INTERCEPT(y,x)

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_INTERCEPT, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_INTERCEPT Window Function

For the REGR\_INTERCEPT window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Intercept of the Regression Line for NbrSold and SalesPrice

This example uses the data from the HomeSales table.

SalesPrice	NbrSold	Area
160000	126	358711030
180000	103	358711030
200000	82	358711030
220000	75	358711030
240000	82	358711030
260000	40	358711030
280000	20	358711030

The following query returns the intercept of the regression line for NbrSold and SalesPrice in the range of 160000 to 280000 in the 358711030 area.

```
SELECT CAST (REGR_INTERCEPT(NbrSold,SalesPrice) AS DECIMAL (5,1))
FROM HomeSales
WHERE area = 358711030
AND SalesPrice BETWEEN 160000 AND 280000;
```

Here is the result:

```
REGR_INTERCEPT(NbrSold,SalesPrice)
```

```
-----  
249.9
```

## Related Topics

For more information, see:

- For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For details on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For the REGR\_INTERCEPT window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## REGR\_R2

### Purpose

Returns the coefficient of determination for all non-null data pairs of the dependent and independent variable arguments.

### Syntax

```
— REGR_R2 — ( dependent_variable_expression, independent_variable_expression ) —————
```

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_R2 can be combined with any of the ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_R2 cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_R2 returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_R2(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE

- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_R2, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_R2 Window Function

For the REGR\_R2 window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Coefficient of Determination for Height and Weight

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
--	-----	-----
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the coefficient of determination for height and weight where neither height nor weight is null.

```
SELECT CAST(REGR_R2(weight,height) AS DECIMAL(4,2))
FROM regrtbl;

REGR_R2(weight,height)
-----
.58
```

## Related Topics

For more information, see:

- For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For the REGR\_R2 window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## REGR\_SLOPE

### Purpose

Returns the slope of the univariate linear regression line through all non-null data pairs of the dependent and independent variable arguments.

### Syntax

— REGR\_SLOPE — ( *dependent\_variable\_expression*, *independent\_variable\_expression* ) —————

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The slope of the best fit linear regression is a measure of the rate of change of the regression of one independent variable on the dependent variable.

The plot of the linear regression on the variables is used to predict the behavior of the dependent variable from the change in the independent variable.

Note that this computation assumes a linear relationship between the variables.

There can be a strong nonlinear relationship between independent and dependent variables, and the computation of the simple linear regression between such variable pairs does not reflect such a relationship.

## Independent and Dependent Variables

An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

A dependent variable is something that is measured in response to a treatment.

For example, you might want to test the ability of various promotions to enhance sales of a particular item.

In this case, the promotion is the independent variable and the sales of the item made as a result of the individual promotion is the dependent variable.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_SLOPE can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_SLOPE cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_SLOPE returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_SLOPE(y, x) are as follows.

Data Type	Format	Title
REAL	Default format of the REAL data type	REGR_SLOPE(y,x)

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_SLOPE, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_SLOPE Window Function

For the REGR\_SLOPE window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## Example: Returning the Slope of the Regression Line for NbrSold and SalesPrice

This example uses the data from the HomeSales table.

SalesPrice	NbrSold	Area
160000	126	358711030
180000	103	358711030
200000	82	358711030
220000	75	358711030

SalesPrice	NbrSold	Area
240000	82	358711030
260000	40	358711030
280000	20	358711030

The following query returns the slope of the regression line for NbrSold and SalesPrice in the range of 160000 to 280000 in the 358711030 area.

```
SELECT CAST (REGR_SLOPE(NbrSold,SalesPrice) AS FLOAT)
FROM HomeSales
WHERE area = 358711030
AND SalesPrice BETWEEN 160000 AND 280000;
```

Here is the result:

```
REGR_SLOPE(NbrSold,SalesPrice)
-----
-7.92857142857143E-004
```

## Related Topics

- For information on the default format of data types and the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For the REGR\_SLOPE window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## REGR\_SXX

### Purpose

Returns the sum of the squares of the *independent\_variable\_expression* for all non-null data pairs of the dependent and independent variable arguments.

### Syntax

```
— REGR_SXX — ( dependent_variable_expression, independent_variable_expression ) —————
```

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_SXX can be combined with any of the ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_SXX cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_SXX returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_SXX(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_SXX, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_SXX Window Function

For the REGR\_SXX window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Sum of Squares for Height

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
--	-----	-----
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150

10	76	?
11	?	150
12	?	?

The following SELECT statement returns the sum of squares for height where neither height nor weight is null.

```
SELECT REGR_SXX(weight,height)
FROM regrtbl;
```

```
Regr_Sxx(weight,height)
-----
                240
```

## Related Topics

For more information, see:

- For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For the REGR\_SXX window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## REGR\_SXY

### Purpose

Returns the sum of the products of the *independent\_variable\_expression* and the *dependent\_variable\_expression* for all non-null data pairs of the dependent and independent variable arguments.

### Syntax

— REGR\_SXY — (*dependent\_variable\_expression*, *independent\_variable\_expression*) —————

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_SXY can be combined with any of the ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_SXY cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_SXY returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_SXY(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE
- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_SXY, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

## REGR\_SXY Window Function

For the REGR\_SXY window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Sum of Products of Height and Weight

This example is based the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
--	-----	-----
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the sum of products of height and weight where neither height nor weight is null.

```
SELECT REGR_SXY(weight,height)
FROM regtbl;

Regr_Sxy(weight,height)
-----
1200
```

## Related Topics

For more information, see *Teradata Vantage™ Data Types and Literals*, B035-1143:

- Information on the default format of data types and an explanation of the formatting characters in the format
- Information on implicit type conversion of UDTs

## REGR\_SYY

### Purpose

Returns the sum of the squares of the *dependent\_variable\_expression* for all non-null data pairs of the dependent and independent variable arguments.

### Syntax

— REGR\_SYY — ( *dependent\_variable\_expression*, *independent\_variable\_expression* ) —

## Syntax Elements

### *dependent\_variable\_expression*

The dependent variable for the regression. A dependent variable is something that is measured in response to a treatment.

The expression cannot contain any ordered analytical or aggregate functions.

### *independent\_variable\_expression*

The independent variable for the regression. An independent variable is a treatment: something that is varied under your control to test the behavior of another variable.

The expression cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Setting Up Axes for Plotting

If you export the data for plotting, define the y-axis (ordinate) as the dependent variable and the x-axis (abscissa) as the independent variable.

## Combination With Other Functions

REGR\_SYY can be combined with any of the ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

REGR\_SYY cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## Computation

When there are fewer than two non-null data point pairs in the data used for the computation, then REGR\_INTERCEPT returns NULL.

Division by zero results in NULL rather than an error.

## Result Type and Attributes

The data type, format, and title for REGR\_SYY(y, x) are as follows.

Data type: REAL

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as y.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Support for UDTs

By default, Teradata Database performs implicit type conversion on UDT arguments that have implicit casts that cast between the UDTs and any of the following predefined types:

- Numeric
- Character
- DATE

- Interval

To define an implicit cast for a UDT, use the CREATE CAST statement and specify the AS ASSIGNMENT clause. For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.

Implicit type conversion of UDTs for system operators and functions, including REGR\_SY, is a Teradata extension to the ANSI SQL standard. To disable this extension, set the DisableUDTImplCastForSysFuncOp field of the DBS Control Record to TRUE. For details, see *Teradata Vantage™ - Database Utilities*, B035-1102.

## REGR\_SY Window Function

For the REGR\_SY window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

### Example: Returning the Sum of Squares for Weight

This example is based on the following regrtbl data. Nulls are indicated by the QUESTION MARK character.

c1	height	weight
1	60	84
2	62	95
3	64	140
4	66	155
5	68	119
6	70	175
7	72	145
8	74	197
9	76	150
10	76	?
11	?	150
12	?	?

The following SELECT statement returns the sum of squares for weight where neither height nor weight is null.

```
SELECT REGR_SY(weight,height)
FROM regrtbl;

Regr_Syy(weight,height)
-----
10426
```

# SKEW

## Purpose

Returns the skewness of the distribution of *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## Syntax

```

— SKEW — ( [ DISTINCT | ALL ] value_expression )

```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

Null and duplicate values specified by *value\_expression* are eliminated from the computation for the group.

### *value\_expression*

A literal or column expression for which the skewness of the distribution of its values is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## Definition

Skewness is the third moment of a distribution. It is a measure of the asymmetry of the distribution about its mean compared with the normal, Gaussian, distribution.

The normal distribution has a skewness of 0.

Positive skewness indicates a distribution having an asymmetric tail extending toward more positive values, while negative skewness indicates an asymmetric tail extending toward more negative values.

## Return Value

This function returns the REAL data type.

## Computation

The equation for computing SKEW is defined as follows:

$$\text{SKEW} = \frac{\text{COUNT}(x)}{(\text{COUNT}(x) - 1)(\text{COUNT}(x) - 2)} \cdot \text{SUM}\left(\frac{x - \text{AVG}(x)}{(\text{STDDEV\_SAMP}(x) ** 3)}\right)$$

where:

This variable ...	Represents ...
x	<i>value_expression</i>

## Conditions That Produce a Null Result

SKEW is valid only for numeric data.

Nulls are not included in the result computation.

The following conditions product a null result:

- Fewer than three non-null data points in the data used for the computation
- $\text{STDDEV\_SAMP}(x) = 0$
- Division by zero

## Related Topics

For more information, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

## STDDEV\_POP

### Purpose

Returns the population standard deviation for the non-null data points in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

## Syntax

```
STDDEV_POP ( { DISTINCT | ALL } value_expression )
```

## Syntax Elements

### ALL

Include all values that are not null specified by *value\_expression*, including duplicates, in the computation. This is the default.

### DISTINCT

To exclude duplicates of *value\_expression* from the computation.

### *value\_expression*

A numeric literal or column expression whose population standard deviation is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The standard deviation is the second moment of a population. For a population, it is a measure of dispersion from the mean of that population.

Do not use STDDEV\_POP unless the data points you are processing are the complete population.

## Combination With Other Functions

STDDEV\_POP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

STDDEV\_POP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## How GROUP BY Affects Report Breaks

STDDEV\_POP operates differently depending on whether there is a GROUP BY clause in the SELECT statement.

IF the query ...	THEN STDDEV_POP is reported for ...
specifies a GROUP BY clause	each individual group.
does not specify a GROUP BY clause	all the rows in the sample.

## Measuring the Standard Deviation of a Population

If your data represents only a sample of the entire population for the variable, then use the STDDEV\_SAMP function. For information, see [STDDEV\\_SAMP](#).

As the sample size increases, the values for STDDEV\_SAMP and STDDEV\_POP approach the same number, but you should always use the more conservative STDDEV\_SAMP calculation unless you are absolutely certain that your data constitutes the entire population for the variable.

## Computation

STANDARD DEVIATION OF A SAMPLE is valid only for numeric data.

Nulls are not included in the result computation.

When there are no non-null data points in the population, then STDDEV\_POP returns NULL.

Division by zero results in NULL rather than an error.

## Return Values

This function returns the REAL data type.

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

## STDDEV\_POP Window Function

For the STDDEV\_POP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## Related Topics

For more information, see:

- *Teradata Vantage™ Data Types and Literals*, B035-1143
- [Window Aggregate Functions](#)

## STDDEV\_SAMP

### Purpose

Returns the sample standard deviation for the non-null data points in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```
STDDEV_SAMP ( DISTINCT  
ALL value_expression )
```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

Exclude duplicates of *value\_expression* from the computation.

### *value\_expression*

A numeric literal or column expression whose sample standard deviation is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The standard deviation is the second moment of a distribution. For a sample, it is a measure of dispersion from the mean of that sample. The computation is more conservative for the population standard deviation to minimize the effect of outliers on the computed value.

## Computation

Division by zero results in NULL rather than an error.

When there are fewer than two non-null data points in the sample used for the computation, then STDDEV\_SAMP returns NULL.

## Return Values

This function returns the REAL data type.

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

## Combination With Other Functions

STDDEV\_SAMP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause. For more information on ordered analytical functions, see [Window Aggregate Functions](#).

STDDEV\_SAMP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## How GROUP BY Affects Report Breaks

The GROUP BY clause affects the STDDEV\_SAMP operation.

IF the query ...	THEN STDDEV_SAMP is reported for ...
specifies a GROUP BY clause	each individual group.
does not specify a GROUP BY clause	all the rows in the sample.

## Measuring the Standard Deviation of a Population

If your data represents the entire population for the variable, then use the STDDEV\_POP function. For information, see [STDDEV\\_POP](#).

As the sample size increases, the values for STDDEV\_SAMP and STDDEV\_POP approach the same number, but you should use the more conservative STDDEV\_SAMP calculation unless you are absolutely certain that your data constitutes the entire population for the variable.

## STDDEV\_SAMP Window Function

For the STDDEV\_SAMP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## Related Topics

For more information, see:

- *Teradata Vantage™ Data Types and Literals*, B035-1143
- [Window Aggregate Functions](#)

## SUM

### Purpose

Returns a column value that is the arithmetic sum of *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

SUM ( DISTINCT  
ALL value_expression )

```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

Exclude duplicate and values that are not null specified by *value\_expression* from the computation.

*value\_expression*

A literal or column expression for which the sum is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Return Values

The following table lists the default attributes for the result of SUM(x).

Data Type of Operand	Data Type of Result	Format	Title
BYTEINT or SMALLINT	INTEGER	Default format of the INTEGER data type	Sum(x)
character	FLOAT	Default format for FLOAT	
UDT	Same as the operand	Format for the data type to which the UDT is implicitly cast	
DECIMAL( <i>n,m</i> )	DECIMAL( <i>p,m</i> ), where <i>p</i> is determined by the rules in the following rules: If MaxDecimal in DBSControl is 0 or 15 and <ul style="list-style-type: none"> <li>• <math>n \leq 15</math>, then <math>p = 15</math>.</li> <li>• <math>15 &lt; n \leq 18</math>, <math>p = 18</math>.</li> <li>• <math>n &gt; 18</math>, then <math>p = 38</math>.</li> </ul> If MaxDecimal in DBSControl is 18 and <ul style="list-style-type: none"> <li>• <math>n \leq 18</math>, then <math>p = 18</math>.</li> <li>• <math>n &gt; 18</math>, then <math>p = 38</math>.</li> </ul> If MaxDecimal in DBSControl is 38 and $n =$ any value, the $p = 38$ .	Default format for the data type of the operand	Sum(x)
Other than UDT, SMALLINT, BYTEINT, DECIMAL, or character	Same as the operand	Default format for the data type of the operand	

## Usage Notes

SUM is valid only for numeric data.

Nulls are not included in the result computation. For details, see “Manipulating Nulls” in *Teradata Vantage™ SQL Fundamentals*, B035-1141 and [Aggregates and Nulls](#).

The SUM function can result in a numeric overflow or the loss of data because of the default output format. If this occurs, a data type declaration may be used to override the default.

For example, if QUANTITY comprises many rows of INTEGER values, it may be necessary to specify a data type declaration like the following for the SUM function:

```
SUM(QUANTITY(FLOAT))
```

## Possible Result Overflow with SELECT Sum

### Possible Result Overflow with SELECT Sum

When using this function, the result can create an overflow when the data type and format are not in sync. For a column defined as:

```
Salary Decimal(15,2) Format '$ZZZ,ZZ9.99'
```

The following query:

```
SELECT SUM (Salary) FROM Employee;
```

causes an overflow because the decimal operand and the format are not in sync.

To avoid possible overflows, explicitly specify the format for decimal sum to specify a format large enough to accommodate the decimal sum resultant data type.

```
SELECT Sum(Salary) (format '$Z,ZZZ,ZZZ,ZZ9.99') FROM Employee;
```

## Examples

### Example: Accounts Receivable

You need to know how much cash you need to pay all vendors who billed you 30 or more days ago.

```
SELECT SUM(Invoice)
FROM AcctsRec
WHERE (CURRENT_DATE - InvDate) >= 30;
```

### Example: Face Value of Inventory

You need to know the total face value for all items in your inventory.

```
SELECT SUM(QUANTITY * Price)
FROM Inventory;
```

```
Sum((QUANTITY * Price))
-----
38,525,151.91
```

## Related Topics

For more information, see:

- For an explanation of the formatting characters in the format, and information on data type default formats, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For the SUM function that returns the cumulative, group, or moving sum, see [Window Aggregate Functions](#).

## UNPIVOT

### Purpose

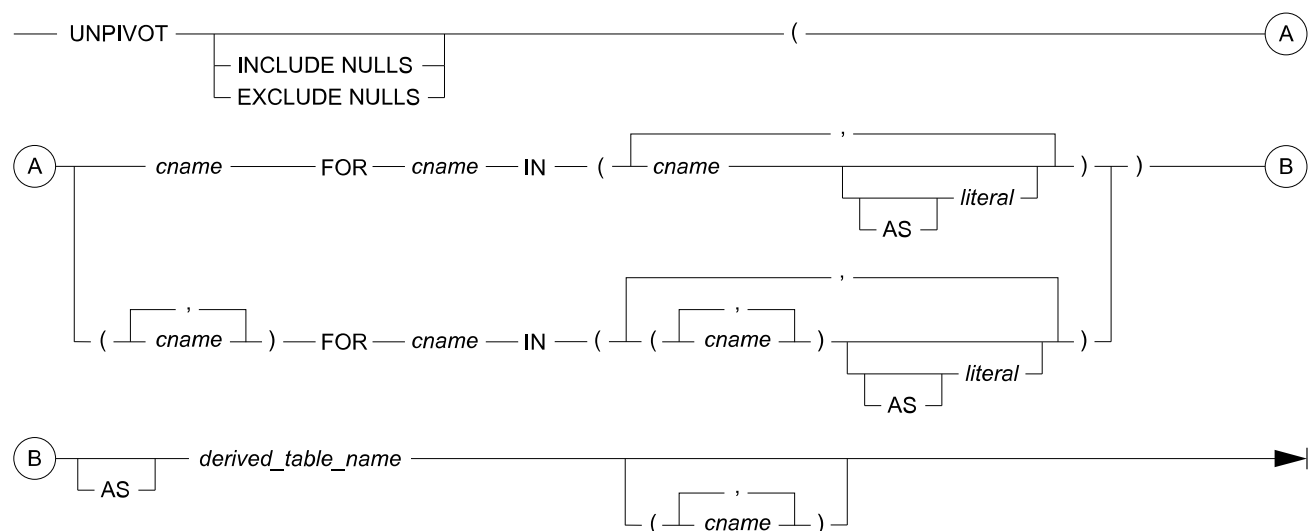
UNPIVOT is the reverse of the PIVOT operation. It provides a mechanism for transforming columns into rows.

The UNPIVOT functionality was introduced previously via the TD\_UNPIVOT table operator. This feature introduces grammar to support the UNPIVOT operator in the FROM clause of the SELECT statement.

### Note:

UNPIVOT invokes the TD\_UNPIVOT table operator internally. You can still use TD\_UNPIVOT independent of UNPIVOT.

### Syntax



## Syntax Elements

### *cname*

A column name.

---

#### **Note:**

For the UNPIVOT operation, column names within the Aggregate functions are referred to as measure columns, and column names in the FOR clause are referred to as pivot columns.

---

### **literal**

Any supported Teradata numeric, character or string literal.

### *derived\_table\_name*

The table name specified for the resultant unpivoted table.

## Usage Notes

---

#### **Note:**

Column names specified just before the FOR clause are referred to as *measure\_columns* in the context of UNPIVOT operation. Column names specified after the FOR clause are referred to as *unpivot\_columns*.

---

Similar to the PIVOT operator, columns with CLOB, BLOB, UDT, XML, or JSON data types are not allowed with the UNPIVOT operator.

The UNPIVOT column name and measure column names cannot be the same as the column names defined in the *derived\_table\_name*.

When multiple *measure\_columns* are involved in UNPIVOT operation, the columns are compatible only if they belong to any of the following three groups:

- CHAR and VARCHAR
- BYTE and VARBYTE
- BYTEINT SMALLINT INTEGER BIGINT REAL DECIMAL NUMBER

Column names specified in the IN list cannot be specified in the assign list of the SELECT statement.

## Examples

The examples in this section use the following denormalized pivoted table, *star1p*, which is defined as:

```
CREATE TABLE star1p(country VARCHAR(20),state VARCHAR(20),Q101Sales
INTEGER,Q201Sales INTEGER,Q301Sales INTEGER,Q101Cogs INTEGER,Q201Cogs
INTEGER,Q301Cogs INTEGER);
```

```
SELECT * FROM star1p;
```

country	state	Q101Sales	Q201Sales	Q301Sales	Q101Cogs	Q201Cogs	Q301Cogs
Canada	ON	?	10	?	?	0	?
Canada	BC	?	?	10	?	?	0
USA	NY	45	?	?	25	?	?
USA	CA	30	50	?	15	20	?

### Example: Unpivoted Sales and Cogs Columns

In this example, the sales and cogs columns are unpivoted.

```
SELECT *
FROM star1p UNPIVOT ((sales,cogs) FOR yr_qtr
                     IN ((Q101Sales, Q101Cogs) AS 'Q101',
                         (Q201Sales, Q201Cogs) AS 'Q201',
                         (Q301Sales, Q301Cogs) AS 'Q301')) Tmp;
```

The output for the unpivoted table:

country	state	yr_qtr	sales	cogs
Canada	ON	Q201	10	0
Canada	ON	Q301	10	0
USA	NY	Q101	45	25
USA	CA	Q101	30	15
USA	CA	Q201	50	20

Note that a pivot combined with a matching unpivot may introduce rows with NULL values. It is possible to unpivot just the 'yr' column.

## Example: Using UNPIVOT for a Unique Year Value

This example shows only one unique value of year, so the unpivot is straightforward.

```
SELECT *
FROM star1p UNPIVOT (Q1sales, Q2sales, Q3sales, Q1cogs, Q2cogs, Q3cogs) FOR
yr IN ((Q101Sales, Q201Sales, Q301Sales, Q101Cogs, Q201Cogs, Q301Cogs) AS
'2001') Tmp;
```

country	state	yr	Q1sales	Q2sales	Q3sales	Q1cogs	Q2cogs	Q3cogs
Canada	ON	2001	?	10	?	?	0	?
Canada	BC	2001	?	?	10	?	?	0
USA	NY	2001	45	?	?	25	?	?
USA	CA	2001	30	50	?	15	20	

## Example: Normalizing the UNPIVOT Operation

This example showcases using UNPIVOT to capture elaborate data of a base table (*star1p*, in this case). The data is spread over many columns into a compact table with an optimal number of columns and no data loss.

```
SELECT *
FROM star1p UNPIVOT (measure_value FOR yr_qtr_measure IN
(Q101Sales, Q201Sales, Q301Sales, Q101Cogs, Q201Cogs, Q301Cogs)) Tmp;
country state yr_qtr_measure measure_value
```

country	state	yr_qtr_measure	measure_value
Canada	BC	Q301Cogs	0
Canada	BC	Q301Sales	10
Canada	ON	Q201Cogs	0
Canada	ON	Q201Sales	10
USA	CA	Q101Cogs	15
USA	CA	Q101Sales	30
USA	CA	Q201Cogs	20
USA	CA	Q201Sales	50
USA	NY	Q101Cogs	25
USA	NY	Q101Sales	45

### Example: Using UNPIVOT with the INCLUDE NULLS Clause

In this example, there are some rows with nulls in the sales and cogs columns. The rows are included in the output when using the INCLUDE NULLS clause.

```
SELECT *
FROM star1p UNPIVOT INCLUDE NULLS ((sales,cogs) FOR yr_qtr IN
((Q101Sales, Q101Cogs) AS 'Q101', (Q201Sales, Q201Cogs) AS 'Q201', (Q301Sales,
Q301Cogs) AS 'Q301')) Tmp;
```

country	state	yr_qtr	sales	cogs
Canada	BC	Q101	?	?
Canada	ON	Q101	?	?
Canada	ON	Q201	10	0
Canada	ON	Q301	10	0
USA	NY	Q101	45	25
USA	CA	Q101	30	15
Canada	BC	Q201	?	?
USA	NY	Q201	?	?
USA	CA	Q201	50	20
Canada	BC	Q301	?	?
USA	NY	Q301	?	?
USA	CA	Q301	?	?

### Example: Using UNPIVOT with the EXCLUDE NULLS Clause

In this example, there are no rows with nulls in either the sales or cogs columns, and the rows are excluded in the output when using EXCLUDE NULLS clause. This is the default option.

```
SELECT *
FROM star1p UNPIVOT EXCLUDE NULLS (sales, cogs) FOR yr_qtr IN
((Q101Sales, Q101Cogs) AS 'Q101', (Q201Sales, Q201Cogs) AS 'Q201', (Q301Sales,
Q301Cogs) AS 'Q301')) Tmp;
```

country	state	yr_qtr	sales	cogs
Canada	ON	Q201	10	0
Canada	ON	Q301	10	0
USA	NY	Q101	45	25
USA	CA	Q101	30	15
USA	CA	Q201	50	20

## Example: Using an IN List with Multiple Column Lists and Unspecified Aliases

In this example, the aliases that the IN list uses were not specified. Instead, the values of the `yr_qtr` column were built by adding the column names with an underscore symbol.

```
SELECT *
FROM star1p UNPIVOT ((sales, cogs) FOR yr_qtr IN
((Q101Sales, Q101Cogs),(Q201Sales, Q201Cogs), (Q301Sales, Q301Cogs)) Tmp;
```

country	state	yr_qtr	sales	cogs
Canada	ON	Q201Sales_Q201Cogs	10	0
Canada	ON	Q301Sales_Q301Cogs	10	0
USA	NY	Q101Sales_Q101Cogs	45	25
USA	CA	Q101Sales_Q101Cogs	30	15
USA	CA	Q201Sales_Q201Cogs	50	20

## Example: Using an IN List that Contains Multiple Columns with a Compatible Data Type

In this example, the `Q101Sales` column contains an `INTEGER` data type, and `Q201Sales` is a `BYTEINT` data type. Both the `INTEGER` and `BYTEINT` data types are compatible with each other.

```
SELECT * FROM star1p UNPIVOT (measure_value FOR yr_qtr_measure IN
(Q101Sales, Q201Sales)) Tmp;
```

country	state	yr_qtr_measure	measure_value
Canada	ON	Q201Sales	10
USA	CA	Q101Sales	30
USA	CA	Q201Sales	50
USA	NY	Q101Sales	45

## Example: Using an IN List that Contains Multiple Columns with an Incompatible Data Type

In this example, the `star1p` table is altered to contain a new column `Q401Sales` with a `VARCHAR(20)` data type. The `Q101Sales` column is an `INTEGER` data type, and the `Q401Sales` is `VARCHAR`.

The `INTEGER` and `VARCHAR` data types are not compatible.

```
SELECT *
FROM star1p UNPIVOT (measure_value FOR yr_qtr_measure IN
(Q101Sales, Q401Sales)) Tmp;
```

Error 9134 Failure in TD\_Unpivot contract function. Error determining column type of value columns.

## Related Topics

For more information, see:

- [PIVOT](#)
- "TD\_UNPIVOT" in *Teradata Vantage™ SQL Operators and User-Defined Functions*, B035-1210

## VAR\_POP

### Purpose

Returns the population variance for the data points in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```
— VAR_POP — ( ( DISTINCT | ALL ) value_expression ) —————
```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

To exclude duplicates of *value\_expression* from the computation.

### *value\_expression*

A numeric literal or column expression whose population variance is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The variance of a population is a measure of dispersion from the mean of that population.

Do not use VAR\_POP unless the data points you are processing are the complete population.

## Computation

When the population has no non-null data points, VAR\_POP returns NULL.

Division by zero results in NULL rather than an error.

## Return Value

This function returns the REAL data type.

## Usage Notes

The following restrictions apply to operands:

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## Combination With Other Functions

VAR\_POP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause.

VAR\_POP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## GROUP BY Affects Report Breaks

The GROUP BY clause affects the VAR\_POP operation.

IF the query ...	THEN VAR_POP is reported for ...
specifies a GROUP BY clause	each individual group.
does not specify a GROUP BY clause	all the rows in the sample.

## Measuring the Standard Deviation of a Population

If your data represents the only a sample of the entire population for the variable, then use the VAR\_SAMP function. For information, see “VAR\_SAMP”.

As the sample size increases, the values for VAR\_SAMP and VAR\_POP approach the same number, but you should always use the more conservative STDDEV\_SAMP calculation unless you are absolutely certain that your data constitutes the entire population for the variable.

## Related Topics

For more information, see:

- *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on ordered analytical functions, see [Overview](#).
- For the VAR\_POP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

## VAR\_SAMP

### Purpose

Returns the sample variance for the data points in *value\_expression*.

To invoke the time series version of this function, use the GROUP BY TIME clause. For more information, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

### Syntax

```

VAR_SAMP ( ( DISTINCT  
ALL ) value_expression )

```

## Syntax Elements

### ALL

All values of *value\_expression* that are not null, including duplicates, are included in the computation.

### DISTINCT

To exclude duplicates of *value\_expression* from the computation.

### *value\_expression*

A numeric literal or column expression whose sample variance is to be computed.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Definition

The variance of a sample is a measure of dispersion from the mean of that sample. It is the square of the sample standard deviation.

The computation is more conservative than that for the population standard deviation to minimize the effect of outliers on the computed value.

## Computation

When the sample used for the computation has fewer than two non-null data points, VAR\_SAMP returns NULL.

Division by zero results in NULL rather than an error.

## Combination With Other Functions

VAR\_SAMP can be combined with ordered analytical functions in a SELECT list, QUALIFY clause, or ORDER BY clause.

VAR\_SAMP cannot be combined with aggregate functions within the same SELECT list, QUALIFY clause, or ORDER BY clause.

## GROUP BY Affects Report Breaks

VAR\_SAMP operates differently depending on whether or not there is a GROUP BY clause in the SELECT statement.

IF the query ...	THEN VAR_SAMP is reported for ...
specifies a GROUP BY clause	each individual group.
does not specify a GROUP BY clause	all the rows in the sample.

## Measuring the Variance of a Population

If your data represents the entire population for the variable, then use the VAR\_POP function.

As the sample size increases, the values for VAR\_SAMP and VAR\_POP approach the same number, but you should always use the more conservative VAR\_SAMP calculation unless you are absolutely certain that your data constitutes the entire population for the variable.

## Return Value

This function returns the REAL data type.

## Usage Notes

For operands:

- If the operand is character, the format is the default format for FLOAT.
- If the operand is numeric, date, or interval, the format is the same format as x.
- If the operand is UDT, the format is the format for the data type to which the UDT is implicitly cast.

The *value\_expression* cannot be a reference to a view column derived from a function, and cannot contain any ordered analytical or aggregate functions.

VARIANCE OF A SAMPLE is valid only for numeric data.

Nulls are not included in the result computation.

Division by zero results in NULL rather than an error.

## Related Topics

For more information, see:

- For more information on ordered analytical functions, see [Overview](#).
- For the VAR\_SAMP window function that performs a group, cumulative, or moving computation, see [Window Aggregate Functions](#).

- If your data represents the entire population for the variable, then use the VAR\_POP function. For information, see [VAR\\_POP](#).
- For more information on CREATE CAST, see *Teradata Vantage™ SQL Data Definition Language Syntax and Examples*, B035-1144.
- For details about the DisableUDTImplCastForSysFuncOp field, see *Teradata Vantage™ - Database Utilities*, B035-1102.
- For more information on implicit type conversion of UDTs, see *Teradata Vantage™ Data Types and Literals*, B035-1143.

# Ordered Analytical/Window Aggregate Functions

## Overview

The following sections describe:

- Ordered analytical functions
- Window Aggregate Functions

## Ordered Analytical Functions

Ordered analytical functions provide support for many common operations in analytical processing and data mining that require an ordered set of results rows or depend on values in a previous row. Ordered analytical functions enable and expedite the processing of queries containing On Line Analytical Processing (OLAP) style decision support requests.

For example, computing a seven-day running sum requires:

- First, that rows be ordered by date.
- Then, that the value for the running sum be computed by:
  - Adding the current row value to the value of the sum from the previous row, and
  - Subtracting the value from the row eight days ago.

## Benefits

Ordered analytical functions extend the Teradata Database query execution engine with the concept of an ordered set and with the ability to use the values from multiple rows in computing a new value.

The result of an ordered analytical function is handled the same as any other SQL expression. It can be a result column or part of a more complex arithmetic expression within its SELECT.

Each of the ordered analytical functions permit you to specify the sort ordering column or columns on which to sort the rows retrieved by the SELECT statement. The sort order and any other input parameters to the functions are specified the same as arguments to other SQL functions and can be any normal SQL expression.

## Ordered Analytical Calculations at the SQL Level

Performing ordered analytical computations at the SQL level rather than through a higher-level OLAP calculation engine provides four distinct advantages.

- Reduced programming effort.
- Elimination of the need for external sort routines.

- Elimination of the need to export large data sets to external tools because ordered analytical functions enable you to target the specific data for analysis within the warehouse itself by specifying conditions in the query.
- Marked enhancement of analysis performance over the slow, single-threaded operations that external tools perform on large data sets.

## Teradata Warehouse Miner

You need not directly code SQL queries to take advantage of ordered analytical functions. Both Teradata Database and many third-party query management and analytical tools have full access to the Teradata SQL ordered analytical functions. Teradata Warehouse Miner, for example, a tool that performs data mining preprocessing inside the database engine, relies on these features to perform functions in the database itself rather than requiring data extraction.

Teradata Warehouse Miner includes approximately 40 predefined data mining functions in SQL based on the Teradata SQL-specific functions. For example, the Teradata Warehouse Miner FREQ function uses the Teradata SQL-specific functions CSUM, RANK, and QUALIFY to determine frequencies.

## Example

The following example shows how the SQL query to calculate a frequency of gender to marital status would appear using Teradata Warehouse Miner.

```
SELECT gender, marital_status, xcnt, xpct
      ,CSUM(xcnt, xcnt DESC, gender, marital_status) AS xcum_cnt
      ,CSUM(xpct, xcnt DESC, gender, marital_status) AS xcum_pct
      ,RANK(xcnt DESC, gender ASC, marital_status ASC) AS xrank
FROM
  (SELECT gender, marital_status, COUNT(*) AS xcnt
    ,100.000 * xcnt / xall (FORMAT 'ZZ9.99') AS xpct
   FROM customer_table A,
    (SELECT COUNT(*) AS xall
     FROM customer_table) B
  GROUP BY gender, marital_status, xall
  HAVING xpct >= 1) T1
QUALIFY xrank <= 8
ORDER BY xcnt DESC, gender, marital_status
```

The result for this query looks like the following table.

gender	marital_status	xcnt	xpct	xcum_cnt	xcum_pct	xrank
F	Married	3910093	36.71	3910093	36.71	1
M	Married	2419511	22.71	6329604	59.42	2
F	Divorced	1612130	15.13	7941734	74.55	3

gender	marital_status	xcnt	xpct	xcum_cnt	xcum_pct	xrank
M	Divorced	1412624	3.26	9354358	87.81	4
F	Single	491224	4.61	9845582	92.42	5
F	Widowed	319881	3.01	10165463	95.43	6
M	Single	319794	3.00	10485257	98.43	7
M	Widowed	197131	1.57	10652388	100.00	8

## Characteristics of Ordered Analytical Functions

### The Function Value

The function value for a column in a row considers that row (and a subset of all other rows in the group) and produces a new value.

The generic function describing this operation is as follows:

```
new_column_value = FUNCTION(column_value, rows_defined_by_window)
```

### Use of QUALIFY Clause

Rows can be eliminated by applying conditions on the new column value. The QUALIFY clause is analogous to the HAVING clause of aggregate functions. The QUALIFY clause eliminates rows based on the function value, returning a new value for each of the participating rows. For example:

```
SELECT StoreID, SUM(profit) OVER (PARTITION BY StoreID)
FROM facts
QUALIFY SUM(profit) OVER (PARTITION BY StoreID) > 2;
```

An SQL query that contains both ordered analytical functions and aggregate functions can have both a QUALIFY clause and a HAVING clause, as in the following example:

```
SELECT StoreID, SUM(sale),
SUM(profit) OVER (PARTITION BY StoreID)
FROM facts
GROUP BY StoreID, sale, profit
HAVING SUM(sale) > 15
QUALIFY SUM(profit) OVER (PARTITION BY StoreID) > 2;
```

### DISTINCT Clause Restriction

The DISTINCT clause is not permitted in window aggregate functions.

## Permitted Query Objects

Ordered analytical functions are permitted in the following database query objects:

- Views
- Macros
- Derived tables
- INSERT ... SELECT

## Where Ordered Analytical Functions are Not Permitted

Ordered analytical functions are not permitted in:

- Subqueries
- WHERE clauses
- SELECT AND CONSUME statements

## Use of Standard SQL Features

You can use standard SQL features within the same query to make your statements more sophisticated.

For example, you can use ordered analytical functions in the following ways.

Use an analytical function in this operation ...	To ...
INSERT ... SELECT	populate a new column.
derived table	create a new table to participate in a complex query.

Ordered analytical functions having different sort expressions are evaluated one after another, reusing the same spool file. Different functions having the same sort expression are evaluated simultaneously.

## Unsupported Data Types

Ordered analytical functions do not operate on the following data types:

- CLOB or BLOB data types
- UDT data types

Note that CLOB, BLOB, or UDT data types are usable inside an expression if the result is a supported data type. For example:

```
SELECT
  RANK() OVER
    (PARTITION BY(CASE WHEN b IS NULL THEN 1 ELSE 0 END) ORDER BY id)
FROM btab;
```

However, the following example results in an error because the function cannot sort by BLOB:

```
SELECT
  RANK() OVER
  (PARTITION BY b ORDER BY id)
FROM btab;
```

## Ordered Analytical Functions and Period Data Types

Expressions that evaluate to Period data types can be specified for any expression within the following ordered analytical functions: QUANTILE, RANK (Teradata-specific function), and RANK (ANSI SQL Window function).

## Ordered Analytical Functions and Recursive Queries

Ordered analytical functions cannot appear in a recursive statement of a recursive query. However, a non-recursive seed statement in a recursive query can specify an ordered analytical function.

## Ordered Analytical Functions and Hash or Join Indexes

When a single table query specifies an ordered analytical function on columns that are also defined for a single table compressed hash or join index, the Optimizer does not select the hash or join index to process the query.

## Ordered Analytical Functions and Row Level Security Tables

When a request that includes an ordered analytical function, such as MAVG, CSUM, or RANK, references a table protected by row level security, the operation is based on only the rows that are accessible to the requesting user. In order to apply all rows of the table to the function, the user must have one of the following:

- The required security credentials to access all rows of the table.
- The required OVERRIDE privileges on the security constraints in the table.

## Computation Sort Order and Result Order

The sort order that you specify in the window specification defines the sort order of the rows over which the function is applied; it does not define the ordering of the results.

For example, to compute the average sales for the months following the current month, order the rows by month:

```
SELECT StoreID, SMonth, ProdID, Sales,
  AVG(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth
                  ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)
FROM sales_tbl;
```

StoreID	SMonth	ProdID	Sales	Remaining Avg(Sales)
-----	-----	-----	-----	-----
1001	6	C	30000.00	?
1001	5	C	30000.00	30000.00
1001	4	C	25000.00	30000.00
1001	3	C	40000.00	28333.33
1001	2	C	25000.00	31250.00
1001	1	C	35000.00	30000.00

The default sort order is ASC for the computation. However, the results are returned in the reverse order.

To order the results, use an ORDER BY phrase in the SELECT statement. For example:

```
SELECT StoreID, SMonth, ProdID, Sales,
AVG(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth
                ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)
FROM sales_tbl
ORDER BY SMonth;
```

StoreID	SMonth	ProdID	Sales	Remaining Avg(Sales)
-----	-----	-----	-----	-----
1001	1	C	35000.00	30000.00
1001	2	C	25000.00	31250.00
1001	3	C	40000.00	28333.33
1001	4	C	25000.00	30000.00
1001	5	C	30000.00	30000.00
1001	6	C	30000.00	?

## Data in Partitioning Column of Window Specification and Resource Impact

The columns specified in the PARTITION BY clause of a window specification determine the partitions over which the ordered analytical function executes. For example, the following query specifies the StoreID column in the PARTITION BY clause to compute the group sales sum for each store:

```
SELECT StoreID, SMonth, ProdID, Sales,
SUM(Sales) OVER (PARTITION BY StoreID)
FROM sales_tbl;
```

At execution time, Teradata Database moves all of the rows that fall into a partition to the same AMP. If a very large number of rows fall into the same partition, the AMP can run out of spool space. For example,

if the `sales_tbl` table in the preceding query has millions or billions of rows, and the `StoreID` column contains only a few distinct values, an enormous number of rows are going to fall into the same partition, potentially resulting in out-of-spool errors.

To avoid this problem, examine the data in the columns of the `PARTITION BY` clause. If necessary, rewrite the query to include additional columns in the `PARTITION BY` clause to create smaller partitions that Teradata Database can distribute more evenly among the AMPs. For example, the preceding query can be rewritten to compute the group sales sum for each store for each month:

```
SELECT StoreID, SMonth, ProdID, Sales,
       SUM(Sales) OVER (PARTITION BY StoreID, SMonth)
FROM sales_tbl;
```

## Using Ordered Analytical Functions

Example: Using `RANK` and `AVG`

Consider the result of the following `SELECT` statement using the following ordered analytical functions, `RANK` and `AVG`.

```
SELECT item, smonth, sales,
       RANK() OVER (PARTITION BY item ORDER BY sales DESC),
       AVG(sales) OVER (PARTITION BY item
                        ORDER BY smonth
                        ROWS 3 PRECEDING)
FROM sales_tbl
ORDER BY item, smonth;
```

The results table might look like the following.

Item	SMonth	Sales	Rank(Sales)	Moving Avg(Sales)
A	1996-01	110	13	110
A	1996-02	130	10	120
A	1996-03	170	6	137
A	1996-04	210	3	155
A	1996-05	270	1	195
A	1996-06	250	2	225
A	1996-07	190	4	230
A	1996-08	180	5	222
A	1996-09	160	7	195
A	1996-10	140	9	168

Item	SMonth	Sales	Rank(Sales)	Moving Avg(Sales)
A	1996-11	150	8	158
A	1996-12	120	11	142
A	1997-01	120	11	132
B	1996-02	30	5	30
...	...	...	...	...

## Example: Using QUALIFY With RANK

Adding a QUALIFY clause to a query eliminates rows from an unqualified table.

For example, if you wanted to see whether the high sales months were unusual, you could add a QUALIFY clause to the previous query.

```
SELECT item, smonth, sales,
       RANK() OVER (PARTITION BY item ORDER BY sales DESC),
       AVG(sales) OVER (PARTITION BY item ORDER BY smonth ROWS 3 PRECEDING)
FROM sales_tbl
ORDER BY item, smonth
QUALIFY RANK() OVER(PARTITION BY item ORDER BY sales DESC) <=5;
```

This additional qualifier produces a results table that might look like the following.

Item	SMonth	Sales	Rank(Sales)	Moving Avg(Sales)
A	1996-04	210	3	155
A	1996-05	270	1	195
A	1996-06	250	2	225
A	1996-07	190	4	230
A	1996-08	180	5	222
B	1996-02	30	1	30
...	...	...	...	...

The result indicates that sales had probably been fairly low prior to the start of the current sales season.

## Example: Using QUALIFY With RANK

Consider the following sales table named sales\_tbl.

Store	ProdID	Sales
1003	C	20000.00
1003	D	50000.00
1003	A	30000.00
1002	C	35000.00
1002	D	25000.00
1002	A	40000.00
1001	C	60000.00
1001	D	35000.00
1001	A	100000.00
1001	B	10000.00

Now perform the following simple SELECT statement against this table, qualifying answer rows by rank.

```
SELECT store, prodID, sales,
RANK() OVER (PARTITION BY store ORDER BY sales DESC)
FROM sales_tbl
QUALIFY RANK() OVER (PARTITION BY store ORDER BY sales DESC) <=3;
```

The result appears in the following typical output table.

Store	ProdID	Sales	Rank(Sales)
1001	A	100000.00	1
1001	C	60000.00	2
1001	D	35000.00	3
1002	A	40000.00	1
1002	C	35000.00	2
1002	D	25000.00	3
1003	D	50000.00	1
1003	A	30000.00	2
1003	C	20000.00	3

Note that every row in the table is returned with the computed value for RANK except those that do not meet the QUALIFY clause (sales rank is less than third within the store).

## Related Topics

For more information, see:

- For more information about row level security, see *Teradata Vantage™ NewSQL Engine Security Administration*, B035-1100.
- For details on the QUALIFY clause, see *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

## The Window Feature

The ANSI SQL:2011 window feature provides a way to dynamically define a subset of data, or *window*, in an ordered relational database table. A window is specified by the OVER() phrase, which can include the following clauses inside the parentheses:

- PARTITION BY
- ORDER BY
- RESET WHEN
- ROWS

## PARTITION BY Phrase

PARTITION BY takes a column reference list and groups the rows based on the specified column reference list over which the ordered analytical function executes. Such a grouping is static. To define a group or partition based on a condition, use the RESET WHEN phrase. For more information, see [RESET WHEN Phrase](#).

If there is no PARTITION BY phrase or RESET WHEN phrase, then the entire result set, delivered by the FROM clause, constitutes a single partition, over which the ordered analytical function executes.

Consider the following table named sales\_tbl.

StoreID	SMonth	ProdID	Sales
1001	1	C	35000.00
1001	2	C	25000.00
1001	3	C	40000.00
1001	4	C	25000.00
1001	5	C	30000.00
1001	6	C	30000.00
1002	1	C	40000.00
1002	2	C	35000.00

StoreID	SMonth	ProdID	Sales
1002	3	C	110000.00
1002	4	C	60000.00
1002	5	C	35000.00
1002	6	C	100000.00

The following SELECT statement, which does not include PARTITION BY, computes the average sales for all the stores in the table:

```
SELECT StoreID, SMonth, ProdID, Sales,
       AVG(Sales) OVER ()
FROM sales_tbl;
```

StoreID	SMonth	ProdID	Sales	Group Avg(Sales)
-----	-----	-----	-----	-----
1001	1	C	35000.00	47083.33
1001	2	C	25000.00	47083.33
1001	3	C	40000.00	47083.33
1001	4	C	25000.00	47083.33
1001	5	C	30000.00	47083.33
1001	6	C	30000.00	47083.33
1002	1	C	40000.00	47083.33
1002	2	C	35000.00	47083.33
1002	3	C	110000.00	47083.33
1002	4	C	60000.00	47083.33
1002	5	C	35000.00	47083.33
1002	6	C	100000.00	47083.33

To compute the average sales for each store, partition the data in sales\_tbl by StoreID:

```
SELECT StoreID, SMonth, ProdID, Sales,
       AVG(Sales) OVER (PARTITION BY StoreID)
FROM sales_tbl;
```

StoreID	SMonth	ProdID	Sales	Group Avg(Sales)
-----	-----	-----	-----	-----
1001	3	C	40000.00	30833.33
1001	5	C	30000.00	30833.33
1001	6	C	30000.00	30833.33
1001	4	C	25000.00	30833.33
1001	2	C	25000.00	30833.33

1001	1	C	35000.00	30833.33
1002	3	C	110000.00	63333.33
1002	5	C	35000.00	63333.33
1002	6	C	100000.00	63333.33
1002	4	C	60000.00	63333.33
1002	2	C	35000.00	63333.33
1002	1	C	40000.00	63333.33

## ORDER BY Phrase

ORDER BY specifies how the rows are ordered in a partition, which determines the sort order of the rows over which the function is applied.

To add the monthly sales for a store in the sales\_tbl table to the sales for previous months, compute the cumulative sales sum and order the rows in each partition by SMonth:

```
SELECT StoreID, SMonth, ProdID, Sales,
       SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sales_tbl;
```

StoreID	SMonth	ProdID	Sales	Cumulative Sum(Sales)
-----	-----	-----	-----	-----
1001	1	C	35000.00	35000.00
1001	2	C	25000.00	60000.00
1001	3	C	40000.00	100000.00
1001	4	C	25000.00	125000.00
1001	5	C	30000.00	155000.00
1001	6	C	30000.00	185000.00
1002	1	C	40000.00	40000.00
1002	2	C	35000.00	75000.00
1002	3	C	110000.00	185000.00
1002	4	C	60000.00	245000.00
1002	5	C	35000.00	280000.00
1002	6	C	100000.00	380000.00

## RESET WHEN Phrase

RESET WHEN is a Teradata extension to the ANSI SQL standard.

Depending on the evaluation of the specified condition, RESET WHEN determines the group or partition, over which the ordered analytical function operates. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition. To define a partition based on a column reference list, use the PARTITION BY phrase. For more information, see [PARTITION BY Phrase](#).

If there is no RESET WHEN phrase or PARTITION BY phrase, then the entire result set, delivered by the FROM clause, constitutes a single partition, over which the ordered analytical function executes.

You can have different RESET WHEN clauses in the same SELECT list.

---

**Note:**

A window specification that specifies a RESET WHEN clause must also specify an ORDER BY clause.

---

## RESET WHEN Condition Rules

The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify conditional partitioning.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

The following rules apply for RESET WHEN conditions.

A RESET WHEN condition can contain the following:

- Ordered analytical functions that do not include the RESET WHEN clause
- Scalar subqueries
- Aggregate operators
- DEFAULT functions

However, DEFAULT without an explicit column specification is valid only if it is specified as a standalone condition in the predicate. For more information, see [Rules For Using a DEFAULT Function As Part of a RESET WHEN Condition](#).

A RESET WHEN condition cannot contain the following:

- Ordered analytical functions that include the RESET WHEN clause
- The SELECT statement
- LOB columns
- UDT expressions, including UDFs that return a UDT value

However, a RESET WHEN condition can include an expression that contains UDTs as long as that expression returns a result that has a predefined data type.

## Rules For Using a DEFAULT Function As Part of a RESET WHEN Condition

The following rules apply to the use of the DEFAULT function as part of a RESET WHEN condition:

- You can specify a DEFAULT function with a column name argument within a predicate. The system evaluates the DEFAULT function to the default value of the column specified as its argument. After the system evaluates the DEFAULT function, it treats it like a literal in the predicate.
- You can specify a DEFAULT function without a column name argument within a predicate only if there is one column specification and one DEFAULT function as the terms on each side of the comparison operator within the expression.
- Following existing comparison rules, a condition with a DEFAULT function used with comparison operators other than IS [NOT] NULL is unknown if the DEFAULT function evaluates to null.

A condition other than IS [NOT]NULL with a DEFAULT function compared with a null evaluates to unknown.

IF a DEFAULT function is used with...	THEN the comparison is...
IS NULL	TRUE if the default is null, else it is FALSE.
IS NOT NULL	FALSE if the default is null, else it is TRUE.

## Examples

### Example

This example finds cumulative sales for all periods of increasing sales for each region.

```
SUM(sales) OVER (
  PARTITION BY region
  ORDER BY day_of_calendar
  RESET WHEN sales < /* preceding row */ SUM(sales) OVER (
    PARTITION BY region
    ORDER BY day_of_calendar
    ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)
  ROWS UNBOUNDED PRECEDING
)
```

### Example

This example finds sequences of increasing balances. This implies that we reset whenever the current balance is less than or equal to the preceding balance.

```
SELECT account_key, month, balance,
  ROW_NUMBER() over
```

```

(PARTITION BY account_key
ORDER BY month
RESET WHEN balance /* current row balance */ <=
SUM(balance) over (PARTITION BY account_key ORDER BY month
ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING) /* prev row */
) - 1 /* to get the count started at 0 */ as balance_increase
FROM accounts;

```

The possible results of the preceding SELECT appear in the table below:

account_key	month	balance	balance_increase
-----	----	-----	-----
1	1	60	0
1	2	99	1
1	3	94	0
1	4	90	0
1	5	80	0
1	6	88	1
1	7	90	2
1	8	92	3
1	9	10	0
1	10	60	1
1	11	80	2
1	12	10	0

## Example

The following example illustrates a window function with a nested aggregate. The query is processed as follows:

1. We use the SUM(balance) aggregate function to calculate the sum of all the balances for a given account in a given quarter.
2. We check to see if a balance in a given quarter (for a given account) is greater than the balance of the previous quarter.
3. If the balance increased, we track a cumulative count value. As long as the RESET WHEN condition evaluates to false, the balance is increasing over successive quarters, and we continue to increase the count.
4. We use the ROW\_NUMBER() ordered analytical function to calculate the count value. When we reach a quarter whose balance is less than or equal to that of the previous quarter, the RESET WHEN condition evaluates to true, and we start a new partition and ROW\_NUMBER() restarts the count from 1. We specify ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING to access the previous value.
5. Finally, we subtract 1 to ensure that the count values start with 0.

The `balance_increase` column shows the number of successive quarters where the balance was increasing. In this example, we only have one quarter (1->2) where the balance has increased.

```
SELECT account_key, quarter, sum(balance),
ROW_NUMBER() over
  (PARTITION BY account_key
   ORDER BY quarter
   RESET WHEN sum(balance) /* current row balance */ <=
    SUM(sum(balance)) over (PARTITION BY account_key ORDER BY quarter
    ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)/* prev row */
   ) - 1 /* to get the count started at 0 */ as balance_increase
FROM accounts
GROUP BY account_key, quarter;
```

The possible results of the preceding SELECT appear in the table below:

account_key	quarter	balance	balance_increase
-----	-----	-----	-----
1	1	253	0
1	2	258	1
1	3	192	0
1	4	150	0

## Example

In the following example, the condition in the `RESET WHEN` clause contains `SELECT` as a nested subquery. This is not allowed and results in an error.

```
SELECT SUM(a1) OVER
  (ORDER BY 1
   RESET WHEN 1 in (SELECT 1))
FROM t1;
$
*** Failure 3706 Syntax error: SELECT clause not supported in
RESET...WHEN clause.
```

## ROWS Phrase

`ROWS` defines the rows over which the aggregate function is computed for each row in the partition.

If `ROWS` is specified, the computation of the aggregate function for each row in the partition includes only the subset of rows in the `ROWS` phrase.

If there is no `ROWS` phrase, then the computation includes all the rows in the partition.

To compute the three-month moving average sales for each store in the sales\_tbl table, partition by StoreID, order by SMonth, and perform the computation over the current row and the two preceding rows:

```
SELECT StoreID, SMonth, ProdID, Sales,
       AVG(Sales) OVER (PARTITION BY StoreID
                        ORDER BY SMonth
                        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sales_tbl;
```

StoreID	SMonth	ProdID	Sales	Moving Avg(Sales)
-----	-----	-----	-----	-----
1001	1	C	35000.00	35000.00
1001	2	C	25000.00	30000.00
1001	3	C	40000.00	33333.33
1001	4	C	25000.00	30000.00
1001	5	C	30000.00	31666.67
1001	6	C	30000.00	28333.33
1002	1	C	40000.00	40000.00
1002	2	C	35000.00	37500.00
1002	3	C	110000.00	61666.67
1002	4	C	60000.00	68333.33
1002	5	C	35000.00	68333.33
1002	6	C	100000.00	65000.00

## Multiple Window Specifications

In an SQL statement using more than one window function, each window function can have a unique window specification.

For example,

```
SELECT StoreID, SMonth, ProdID, Sales,
       AVG(Sales) OVER (PARTITION BY StoreID
                        ORDER BY SMonth
                        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),
       RANK() OVER (PARTITION BY StoreID ORDER BY Sales DESC)
FROM sales_tbl;
```

## Related Topics

For more information, see:

- See [DEFAULT](#) for more information about the DEFAULT function.

- The window specification can also be applied to a user-defined aggregate function. For details, see [SQL UDF](#).
- To see the syntax for the OVER() phrase and the associated clauses, see [Window Aggregate Functions](#).

## Window Aggregate Functions

An aggregate function on which a window specification is applied is called a window aggregate function. Without a window specification, aggregate functions return one value for all qualified rows examined. Window aggregate functions return a new value for each of the qualifying rows participating in the query. Thus, the following SELECT statement, which includes the aggregate AVG, returns one value only: the average of sales.

```
SELECT AVG(sale)
FROM monthly_sales;

Average(sale)
-----
          1368
```

The AVG window function retains each qualifying row.

The following SELECT statement might return the results that follow.

```
SELECT territory, smonth, sales,
AVG(sales) OVER (PARTITION BY territory
                  ORDER BY smonth ROWS 2 PRECEDING)
FROM sales_history;
```

territory	smonth	sales	Moving Avg(sales)
-----	-----	-----	-----
East	199810	10	10
East	199811	4	7
East	199812	10	8
East	199901	7	7
East	199902	10	9
West	199810	8	8
West	199811	12	10
West	199812	7	9
West	199901	11	10
West	199902	6	8

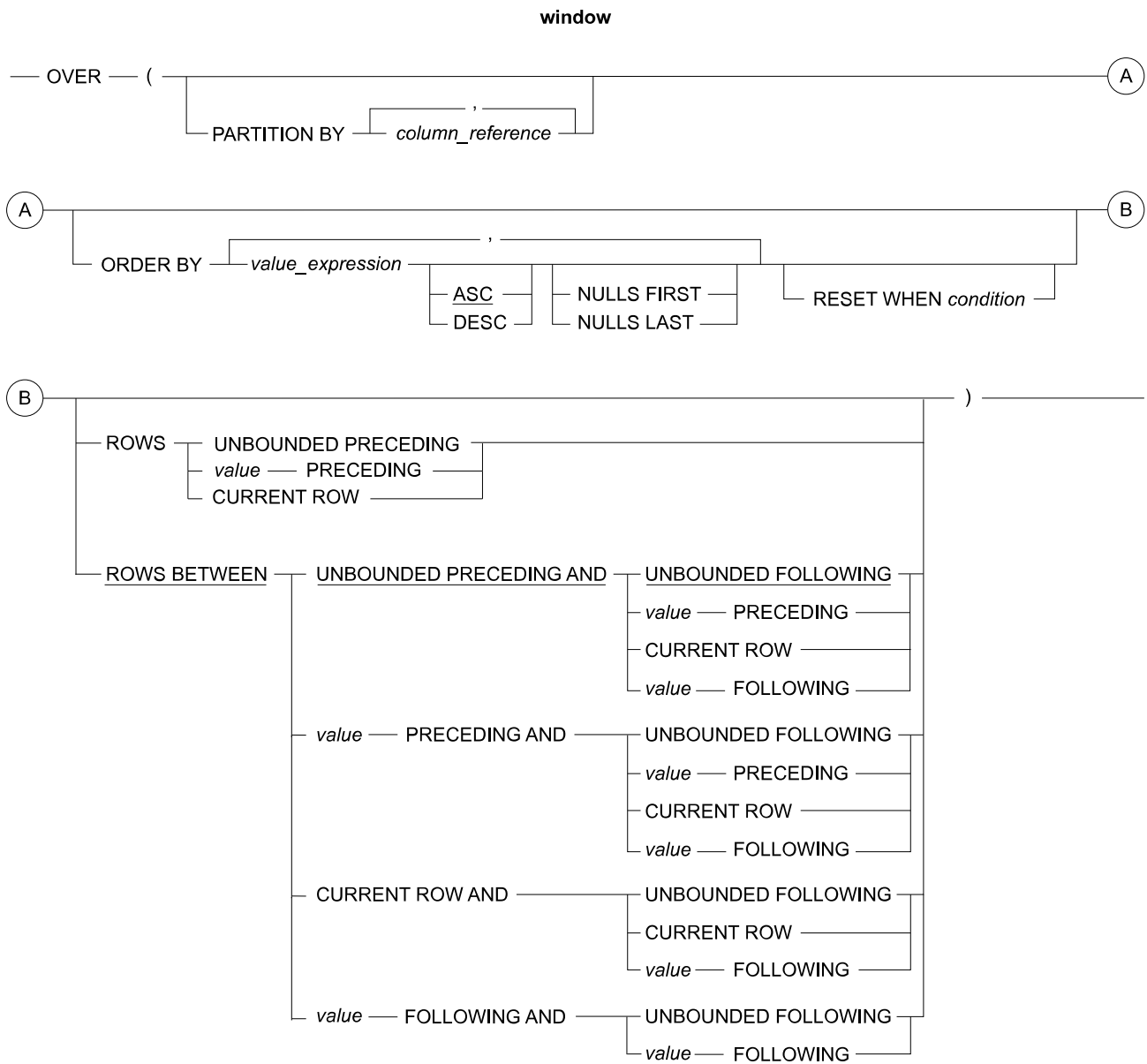
## The Window Specification

### Purpose

Cumulative, group, moving, or remaining computation of an aggregate function.

### Syntax

AVG	— (value_expression)	A
CORR	— (value_expression_1, value_expression_2)	
COUNT	— ( $\underbrace{\text{value\_expression}}_{*}$ )	
COVAR_POP	— (value_expression_1, value_expression_2)	
COVAR_SAMP	— (value_expression_1, value_expression_2)	
MAX	— (value_expression)	
MIN	— (value_expression)	
REGR_AVGX	— (dependent_variable_expression_independentvariable_expression)	
REGR_AVGY	— (dependent_variable_expression_independentvariable_expression)	
REGR_COUNT	— (dependent_variable_expression_independentvariable_expression)	
REGR_INTERCEPT	— (dependent_variable_expression_independentvariable_expression)	
REGR_R2	— (dependent_variable_expression_independentvariable_expression)	
REGR_SLOPE	— (dependent_variable_expression_independentvariable_expression)	
REGR_SXX	— (dependent_variable_expression_independentvariable_expression)	
REGR_SXY	— (dependent_variable_expression_independentvariable_expression)	
REGR_SYY	— (dependent_variable_expression_independentvariable_expression)	
STDDEV_POP	— (value_expression)	
STDDEV_SAMP	— (value_expression)	
SUM	— (value_expression)	
VAR_POP	— (value_expression)	
VAR_SAMP	— (value_expression)	
A —————   window   —————▶		



## Syntax Elements

**ASC**

That the results are to be ordered in ascending sort order.

If the sort field is a character string, the system orders it in ascending order according to the definition of the collation sequence for the current session.

The default order is ASC.

## OVER

How values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the PARTITION BY and RESET WHEN clauses, sorted according to the ORDER BY clause, and considered according to the aggregation group within the partition.

## PARTITION BY

In its *column\_reference*, or comma-separated list of column references, the group, or groups, over which the function operates.

PARTITION BY is optional. If there are no PARTITION BY or RESET WHEN clauses, then the entire result set, delivered by the FROM clause, constitutes a single group, or partition.

PARTITION BY clause is also called the window partition clause.

## ORDER BY

In its *value\_expression* the order in which the values in a group, or partition, are sorted.

## DESC

That the results are to be ordered in descending sort order.

If the sort field is a character string, the system orders it in descending order according to the definition of the collation sequence for the current session.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group or partition, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

RESET WHEN is optional. If there are no RESET WHEN or PARTITION BY clauses, then the entire result set, delivered by the FROM clause, constitutes a single partition.

If RESET WHEN is specified, then the ORDER BY clause must be specified also.

### *condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

## **ROWS**

the starting point for the aggregation group within the partition. The aggregation group end is the current row.

The aggregation group of a row R is a set of rows, defined relative to R in the ordering of the rows within the partition.

If there are no ROWS or ROWS BETWEEN clause, the default aggregation group is ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

The default when there is no ROWS clause for FIRST\_VALUE/LAST\_VALUE is different. For more information, see [FIRST\\_VALUE / LAST\\_VALUE](#).

## **ROWS BETWEEN**

The aggregation group start and end, which defines a set of rows relative to the current row in the ordering of the rows within the partition.

The row specified by the group start must precede the row specified by the group end.

If there are no ROWS or ROWS BETWEEN clause, the default aggregation group is ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

## **UNBOUNDED PRECEDING**

The entire partition preceding the current row.

## UNBOUNDED FOLLOWING

The entire partition following the current row.

## CURRENT ROW

The start or end of the aggregation group as the current row.

## *value* PRECEDING

The number of rows preceding the current row.

The value for *value* is always a positive integer literal.

The maximum number of rows in an aggregation group is 4096 when *value* PRECEDING appears as the group start or group end.

## *value* FOLLOWING

The number of rows following the current row.

The value for *value* is always a positive integer literal.

The maximum number of rows in an aggregation group is 4096 when *value* FOLLOWING appears as the group start or group end.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant, but includes non-ANSI Teradata Database extensions.

In the presence of an ORDER BY clause and the absence of a ROWS or ROWS BETWEEN clause, ANSI SQL:2011 window aggregate functions use ROWS UNBOUNDED PRECEDING as the default aggregation group, whereas Teradata SQL window aggregate functions use ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

## Type of Computation

To compute this type of function ...	Use this aggregation group ...
Cumulative	<ul style="list-style-type: none"> <li>• ROWS UNBOUNDED PRECEDING</li> <li>• ROWS BETWEEN UNBOUNDED PRECEDING AND <i>value</i> PRECEDING</li> <li>• ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW</li> <li>• ROWS BETWEEN UNBOUNDED PRECEDING AND <i>value</i> FOLLOWING</li> </ul>

To compute this type of function ...	Use this aggregation group ...
Group	ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
Moving	<ul style="list-style-type: none"> <li>• ROWS <i>value</i> PRECEDING</li> <li>• ROWS CURRENT ROW</li> <li>• ROWS BETWEEN <i>value</i> PRECEDING AND <i>value</i> PRECEDING</li> <li>• ROWS BETWEEN <i>value</i> PRECEDING AND CURRENT ROW</li> <li>• ROWS BETWEEN <i>value</i> PRECEDING AND <i>value</i> FOLLOWING</li> <li>• ROWS BETWEEN CURRENT ROW AND CURRENT ROW</li> <li>• ROWS BETWEEN CURRENT ROW AND <i>value</i> FOLLOWING</li> <li>• ROWS BETWEEN <i>value</i> FOLLOWING AND <i>value</i> FOLLOWING</li> </ul>
Remaining	<ul style="list-style-type: none"> <li>• ROWS BETWEEN <i>value</i> PRECEDING AND UNBOUNDED FOLLOWING</li> <li>• ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING</li> <li>• ROWS BETWEEN <i>value</i> FOLLOWING AND UNBOUNDED FOLLOWING</li> </ul>

## Arguments to Window Aggregate Functions

Window aggregate functions can take literals, literal expressions, column names (sales, for example), or column expressions (sales + profit) as arguments.

Window aggregates can also take regular aggregates as input parameters to the PARTITION BY and ORDER BY clauses. The RESET WHEN clause can take an aggregate as part of the RESET WHEN condition clause.

COUNT can take “\*” as an input argument, as in the following SQL query:

```
SELECT city, kind, sales, profit,
COUNT(*) OVER (PARTITION BY city, kind
                 ROWS BETWEEN UNBOUNDED PRECEDING AND
                 UNBOUNDED FOLLOWING)
FROM activity_month;
```

## Result Type and Format

The result data type and format for window aggregate functions are as follows.

Function	Result Type	Format
AVG(x) where x is a character type	FLOAT	Default format for FLOAT
AVG(x)	FLOAT	Same format as operand x

Function	Result Type	Format
where x is a numeric, DATE, or INTERVAL type		
CORR(x,y) COVAR_POP(x,y) COVAR_SAMP(x,y) REGR_AVGX( y,x) REGR_AVGY( y,x) REGR_INTERCEPT(x,y) REGR_R2(x,y) REGR_SLOPE(x,y) REGR_SXX(x,y) REGR_SXY(x,y) REGR_SYY(x,y) STDDEV_POP(x,) STDDEV_SAMP(x,) VAR_POP(x,) VAR_SAMP(x) where x is a character type	FLOAT	Default format for FLOAT
CORR(x,y) COVAR_POP(x,y) COVAR_SAMP(x,y) REGR_AVGX( y,x) REGR_AVGY( y,x) REGR_INTERCEPT(x,y) REGR_R2(x,y) REGR_SLOPE(x,y) REGR_SXX(x,y) REGR_SXY(x,y) REGR_SYY(x,y) STDDEV_POP(x) STDDEV_SAMP(x) VAR_POP(x) VAR_SAMP(x) where x is one of the following types: <ul style="list-style-type: none"> <li>• Numeric</li> <li>• DATE</li> <li>• Interval</li> </ul>	Same data type as operand x.	Default format for the data type of operand x
REGR_AVGX(y,x) REGR_AVGY(y, x) where x is a UDT		Default format for the data type to which the UDT is implicitly cast.
COUNT(x)	If MaxDecimal in DBSControl is...	

Function	Result Type	Format
COUNT(*) REGR_COUNT(x,y) where the transaction mode is ANSI	<ul style="list-style-type: none"> <li>0 or 15, then the result type is DECIMAL(15,0) and the format is -(15)9.</li> <li>18, then the result type is DECIMAL(18,0) and the format is -(18)9.</li> <li>38, then the result type is DECIMAL(38,0) and the format is -(38)9.</li> </ul> ANSI transaction mode uses DECIMAL because tables frequently have a cardinality exceeding the range of INTEGER.	
COUNT(x) COUNT(*) REGR_COUNT(x,y) where the transaction mode is Teradata	INTEGER Teradata transaction mode uses INTEGER to avoid regression problems.  <b>Note:</b> You can cast the final result of a COUNT window aggregate function; however, the cast is not used as part of the window function computation as it is for the COUNT aggregate function and, therefore, cannot be used to avoid numeric overflow errors that might occur during the computation.	Default format for INTEGER
MAX(x), MIN(x)	Same data type as operand x.	Same format as operand x
SUM(x) where x is a character type	Same as operand x.	Default format for FLOAT
SUM(x) where x is a DECIMAL(n,m) type	DECIMAL(p,m), where p is determined according to the following rules: If MaxDecimal in DBSControl is 0 or 15 and <ul style="list-style-type: none"> <li><math>n \leq 15</math>, then <math>p = 15</math>.</li> <li><math>15 &lt; n \leq 18</math>, <math>p = 18</math>.</li> <li><math>n &gt; 18</math>, then <math>p = 38</math>.</li> </ul> If MaxDecimal in DBSControl is 18 and <ul style="list-style-type: none"> <li><math>n \leq 18</math>, then <math>p = 18</math>.</li> <li><math>n &gt; 18</math>, then <math>p = 38</math>.</li> </ul> If MaxDecimal in DBSControl is 38 and $n = \text{any value}$ , the $p = 38$ .	Default format for DECIMAL
SUM(x) where x is any numeric type other than DECIMAL	Same as operand x.	Default format for the data type of the operand

## Result Title

The default title that appears in the heading for displayed or printed results depends on the type of computation performed.

IF the type of computation is ...	THEN the result title is ...
cumulative	<p>Cumulative <i>Function_name (argument_list)</i>            For example, consider the following computation:</p> <pre>SELECT AVG(sales) OVER (PARTITION BY region                         ORDER BY smonth ROWS UNBOUNDED PRECEDING) FROM sales_history;</pre> <p>The title that appears in the result heading is:            Cumulative Avg(sales)</p>
group	<p>Group <i>Function_name (argument_list)</i>            For example, consider the following computation:</p> <pre>SELECT AVG(sales) OVER (PARTITION BY region                         ORDER BY smonth ROWS BETWEEN UNBOUNDED                         PRECEDING AND UNBOUNDED FOLLOWING) FROM sales_history;</pre> <p>The title that appears in the result heading is:            Group Avg(sales)</p>
moving	<p>Moving <i>Function_name (argument_list)</i>            For example, consider the following computation:</p> <pre>SELECT AVG(sales) OVER (PARTITION BY region                         ORDER BY smonth ROWS 2 PRECEDING) FROM sales_history;</pre> <p>The title that appears in the result heading is:            Moving Avg(sales)</p>
remaining	<p>Remaining <i>Function_name (argument_list)</i>            For example, consider the following computation:</p> <pre>SELECT AVG(sales) OVER (PARTITION BY region                         ORDER BY smonth ROWS BETWEEN CURRENT ROW                         AND UNBOUNDED FOLLOWING) FROM sales_history;</pre> <p>The title that appears in the result heading is:            Remaining Avg(sales)</p>

## Problems with Missing Data

Make sure that data you analyze has no missing data points. Computing a moving function over data with missing points produces unexpected and incorrect results because the computation considers  $n$  physical rows of data rather than  $n$  logical data points.

## Nesting Aggregates in Window Functions

Although you can nest aggregates in window functions, including the select list, HAVING, QUALIFY, and ORDER BY clauses, the HAVING clause can only contain aggregate function references because HAVING cannot contain nested syntax like RANK() OVER (ORDER BY SUM(x)).

Aggregate functions cannot be specified with Teradata-specific functions.

### Example

The following query nests the SUM aggregate function within the RANK ordered analytical function in the select list:

```
SELECT state, city, SUM(sale),
       RANK() OVER (PARTITION BY state ORDER BY SUM(sale))
FROM T1
WHERE T1.cityID = T2.cityID
GROUP BY state, city
HAVING MAX(sale) > 10;
```

## Alternative: Using Derived Tables

Although only window functions allow aggregates specified together in the same SELECT list, window functions and Teradata-specific functions can be combined with aggregates using derived tables or views. Using derived tables or views also clarifies the semantics of the computation.

### Example

The following example shows the sales rank of a particular product in a store and its percent contribution to the store sales for the top three products in each store.

```
SELECT RT.storeid, RT.prodid, RT.sales,
       RT.rank_sales, RT.sales * 100.0/ST.sum_store_sales
FROM (SELECT storeid, prodid, sales, RANK(sales) AS rank_sales
      FROM sales_tbl
      GROUP BY storeID
      QUALIFY RANK(sales) <=3) AS RT,
     (SELECT storeID, SUM(sales) AS sum_store_sales
      FROM sales_tbl
```

```
GROUP BY storeID) AS ST
WHERE RT.storeID = ST.storeID
ORDER BY RT.storeID, RT.sales;
```

The results table might look something like the following.

storeID	prodID	sales	rank_sales	sales*100.0/sum_store_sales
1001	D	35000.00	3	17.949
1001	C	60000.00	2	30.769
1001	A	100000.00	1	51.282
1002	D	25000.00	3	25.000
1002	C	35000.00	2	35.000
1002	A	40000.00	1	40.000
1003	C	20000.00	3	20.000
1003	A	30000.00	2	30.000
1003	D	50000.00	1	50.000
...	...	...	...	

## Teradata-Specific Alternatives to Ordered Analytical Functions

Teradata SQL supports two syntax alternatives for ordered analytical functions:

- Teradata-specific
- ANSI SQL:2011 compliant

Window aggregate, rank, distribution, and row number functions are ANSI SQL:2011 compliant. Teradata-specific functions are not.

## Teradata-Specific Functions and ANSI SQL:2011 Window Functions

The following table identifies equivalent ANSI SQL:2011 window functions for Teradata-specific functions.

### Note:

The use of the Teradata-specific functions listed in the following table is strongly discouraged. These functions are retained only for backward compatibility with existing applications. Be sure to use the ANSI-compliant window functions for any new applications you develop.

Teradata-Specific Functions	Equivalent ANSI SQL:2011 Window Functions
CSUM	SUM
MAVG	AVG
MDIFF(x, w, y)	composable from SUM
MLINREG	composable from SUM and COUNT
QUANTILE	composable from RANK and COUNT
RANK	RANK
MSUM	SUM

## Comparing Window Aggregate Functions and Teradata-Specific Functions

Avoid using Teradata-specific functions such as MAVG, CSUM, and MSUM for applications intended to be ANSI-compliant and portable.

ANSI Function	Teradata Function	Relationship
AVG	MAVG	<p>The form of the AVG window function that specifies an aggregation group of ROWS <i>value</i> PRECEDING is the ANSI equivalent of the MAVG Teradata-specific function.</p> <p>Note that the ROWS <i>value</i> PRECEDING phrase specifies the number of rows preceding the current row that are used, together with the current row, to compute the moving average. The total number of rows in the aggregation group is <i>value</i> + 1. For the MAVG function, the total number of rows in the aggregation group is the value of <i>width</i>.</p> <p>For AVG window function, an aggregation group of ROWS 5 PRECEDING, for example, means that the 5 rows preceding the current row, plus the current row, are used to compute the moving average. Thus the moving average for the 6th row of a partition would have considered row 6, plus rows 5, 4, 3, 2, and 1 (that is, 6 rows in all).</p> <p>For the MAVG function, a <i>width</i> of 5 means that the current row, plus 4 preceding rows, are used to compute the moving average. The moving average for the 6th row would have considered row 6, plus rows 4, 5, 3, and 2 (that is, 5 rows in all).</p>
SUM	CSUM MSUM	<p>Be sure to use the ANSI-compliant SUM window function for any new applications you develop. Avoid using CSUM and MSUM for applications intended to be ANSI-compliant and portable.</p> <p>The following defines the relationship between the SUM window function and the CSUM and MSUM Teradata-specific functions, respectively:</p> <ul style="list-style-type: none"> <li>• The SUM window function that uses the ORDER BY clause and specifies ROWS UNBOUNDED PRECEDING is the ANSI equivalent of CSUM.</li> <li>• The SUM window function that uses the ORDER BY clause and specifies ROWS <i>value</i> PRECEDING is the ANSI equivalent of MSUM.</li> </ul>

ANSI Function	Teradata Function	Relationship
		<p>Note that the ROWS <i>value</i> PRECEDING phrase specifies the number of rows preceding the current row that are used, together with the current row, to compute the moving average. The total number of rows in the aggregation group is <i>value</i> + 1. For the MSUM function, the total number of rows in the aggregation group is the value of <i>width</i>.</p> <p>Thus for the SUM window function that computes a moving sum, an aggregation group of ROWS 5 PRECEDING means that the 5 rows preceding the current row, plus the current row, are used to compute the moving sum. The moving sum for the 6th row of a partition, for example, would have considered row 6, plus rows 5, 4, 3, 2, and 1 (that is, 6 rows in all).</p> <p>For the MSUM function, a <i>width</i> of 5 means that the current row, plus 4 preceding rows, are used to compute the moving sum. The moving sum for the 6th row, for example, would have considered row 6, plus rows 5, 4, 3, and 2 (that is, 5 rows in all).</p> <p>Moreover, for data having fewer than <i>width</i> rows, MSUM computes the sum using all the preceding rows. MSUM returns the current sum rather than nulls when the number of rows in the sample is fewer than <i>width</i>.</p>

### Example: Group Count

The following SQL query might yield the results that follow it, where the group count for sales is returned for each of the four partitions defined by city and kind. Notice that rows that have no sales are not counted.

```
SELECT city, kind, sales, profit,
COUNT(sales) OVER (PARTITION BY city, kind
                     ROWS BETWEEN UNBOUNDED PRECEDING AND
                     UNBOUNDED FOLLOWING)
FROM activity_month;
```

city	kind	sales	profit	Group Count(sales)
-----	-----	-----	-----	-----
LA	Canvas	45	320	4
LA	Canvas	125	190	4
LA	Canvas	125	400	4
LA	Canvas	20	120	4
LA	Leather	20	40	1
LA	Leather	?	?	1
Seattle	Canvas	15	30	3
Seattle	Canvas	20	30	3
Seattle	Canvas	20	100	3
Seattle	Leather	35	50	1
Seattle	Leather	?	?	1

**Example: Remaining Count**

To count all the rows, including rows that have no sales, use COUNT(\*). Here is an example that counts the number of rows remaining in the partition after the current row:

```
SELECT city, kind, sales, profit,
COUNT(*) OVER (PARTITION BY city, kind ORDER BY profit DESC
                ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)
FROM activity_month;
```

city	kind	sales	profit	Remaining Count(*)
-----	-----	-----	-----	-----
LA	Canvas	20	120	?
LA	Canvas	125	190	1
LA	Canvas	45	320	2
LA	Canvas	125	400	3
LA	Leather	?	?	?
LA	Leather	20	40	1
Seattle	Canvas	15	30	?
Seattle	Canvas	20	30	1
Seattle	Canvas	20	100	2
Seattle	Leather	?	?	?
Seattle	Leather	35	50	1

Note that the sort order that you specify in the window specification defines the sort order of the rows over which the function is applied; it does not define the ordering of the results.

In the example, the DESC sort order is specified for the computation, but the results are returned in the reverse order.

To order the results, use the ORDER BY phrase in the SELECT statement:

```
SELECT city, kind, sales, profit,
COUNT(*) OVER (PARTITION BY city, kind ORDER BY profit DESC
                ROWS BETWEEN 1 FOLLOWING AND
                UNBOUNDED FOLLOWING)
FROM activity_month
ORDER BY city, kind, profit DESC;
```

city	kind	sales	profit	Remaining Count(*)
-----	-----	-----	-----	-----
LA	Canvas	125	400	3
LA	Canvas	45	320	2
LA	Canvas	125	190	1
LA	Canvas	20	120	?

LA	Leather	20	40	1
LA	Leather	?	?	?
Seattle	Canvas	20	100	2
Seattle	Canvas	20	30	1
Seattle	Canvas	15	30	?
Seattle	Leather	35	50	1
Seattle	Leather	?	?	?

### Example: Cumulative Maximum

The following SQL query might yield the results that follow it, where the cumulative maximum value for sales is returned for each partition defined by city and kind.

```
SELECT city, kind, sales, week,
       MAX(sales) OVER (PARTITION BY city, kind
                        ORDER BY week ROWS UNBOUNDED PRECEDING)
FROM activity_month;
```

city	kind	sales	week	Cumulative Max(sales)
-----	-----	-----	----	-----
LA	Canvas	263	16	263
LA	Canvas	294	17	294
LA	Canvas	321	18	321
LA	Canvas	274	20	321
LA	Leather	144	16	144
LA	Leather	826	17	826
LA	Leather	489	20	826
LA	Leather	555	21	826
Seattle	Canvas	100	16	100
Seattle	Canvas	182	17	182
Seattle	Canvas	94	18	182
Seattle	Leather	933	16	933
Seattle	Leather	840	17	933
Seattle	Leather	899	18	933
Seattle	Leather	915	19	933
Seattle	Leather	462	20	933

### Example: Cumulative Minimum

The following SQL query might yield the results that follow it, where the cumulative minimum value for sales is returned for each partition defined by city and kind.

```

SELECT city, kind, sales, week,
MIN(sales) OVER (PARTITION BY city, kind
                  ORDER BY week
                  ROWS UNBOUNDED PRECEDING)
FROM activity_month;

```

city	kind	sales	week	Cumulative Min(sales)
-----	-----	-----	----	-----
LA	Canvas	263	16	263
LA	Canvas	294	17	263
LA	Canvas	321	18	263
LA	Canvas	274	20	263
LA	Leather	144	16	144
LA	Leather	826	17	144
LA	Leather	489	20	144
LA	Leather	555	21	144
Seattle	Canvas	100	16	100
Seattle	Canvas	182	17	100
Seattle	Canvas	94	18	94
Seattle	Leather	933	16	933
Seattle	Leather	840	17	840
Seattle	Leather	899	18	840
Seattle	Leather	915	19	840
Seattle	Leather	462	20	462

### Example: Cumulative Sum

The following query returns the cumulative balance per account ordered by transaction date:

```

SELECT acct_number, trans_date, trans_amount,
SUM(trans_amount) OVER (PARTITION BY acct_number
                        ORDER BY trans_date
                        ROWS UNBOUNDED PRECEDING) as balance
FROM ledger
ORDER BY acct_number, trans_date;

```

Here are the possible results of the preceding SELECT.

acct_number	trans_date	trans_amount	balance
73829	1998-11-01	113.45	113.45
73829	1988-11-05	-52.01	61.44
73929	1998-11-13	36.25	97.69

acct_number	trans_date	trans_amount	balance
82930	1998-11-01	10.56	10.56
82930	1998-11-21	32.55	43.11
82930	1998-11-29	-5.02	38.09

### Example: Group Sum

The query below finds the total sum of meat sales for each city.

```
SELECT city, kind, sales,
       SUM(sales) OVER (PARTITION BY city ROWS BETWEEN UNBOUNDED PRECEDING
                        AND UNBOUNDED FOLLOWING) FROM monthly;
```

The possible results of the preceding SELECT appear in the following table.

city	kind	sales	Group Sum (sales)
Omaha	pure pork	45	220
Omaha	pure pork	125	220
Omaha	pure pork	25	220
Omaha	variety pack	25	220
Chicago	variety pack	55	175
Chicago	variety pack	45	175
Chicago	pure pork	50	175
Chicago	variety pack	25	175

### Example: Group Sum

The following query returns the total sum of meat sales for all cities. Note there is no PARTITION BY clause in the SUM function, so all cities are included in the group sum.

```
SELECT city, kind, sales,
       SUM(sales) OVER (ROWS BETWEEN UNBOUNDED PRECEDING AND
                        UNBOUNDED FOLLOWING)
FROM monthly;
```

The possible results of the preceding SELECT appear in the table below.

city	kind	sales	Group Sum (sales)
Omaha	pure pork	45	395
Omaha	pure pork	125	395
Omaha	pure pork	25	395
Omaha	variety pack	25	395
Chicago	variety pack	55	395
Chicago	variety pack	45	395
Chicago	pure pork	50	395
Chicago	variety pack	25	395

### Example: Moving Sum

The following query returns the moving sum of meat sales by city. Notice that the query returns the moving sum of sales by city (the partition) for the current row (of the partition) and three preceding rows where possible.

The order in which each meat variety is returned is the default ascending order according to profit.

Where no sales figures are available, no moving sum of sales is possible. In this case, there is a null in the sum(sales) column.

```
SELECT city, kind, sales, profit,
       SUM(sales) OVER (PARTITION BY city, kind
                       ORDER BY profit ROWS 3 PRECEDING)
FROM monthly;
```

city	kind	sales	profit	Moving sum (sales)
Omaha	pure pork	25	40	25
Omaha	pure pork	25	120	50
Omaha	pure pork	45	140	95
Omaha	pure pork	125	190	220
Omaha	pure pork	45	320	240
Omaha	pure pork	1255	400	340
Omaha	variety pack	?	?	?
Omaha	variety pack	25	40	25
Omaha	variety pack	25	120	50

city	kind	sales	profit	Moving sum (sales)
Chicago	pure pork	?	?	?
Chicago	pure pork	15	10	15
Chicago	pure pork	54	12	69
Chicago	pure pork	14	20	83
Chicago	pure pork	54	24	137
Chicago	pure pork	14	34	136
Chicago	pure pork	95	80	177
Chicago	pure pork	95	140	258
Chicago	pure pork	15	220	219
Chicago	variety pack	23	39	23
Chicago	variety pack	25	40	48
Chicago	variety pack	125	70	173
Chicago	variety pack	125	100	298
Chicago	variety pack	23	100	298
Chicago	variety pack	25	120	298

### Example: Remaining Sum

The following query returns the remaining sum of meat sales for all cities. Note there is no PARTITION BY clause in the SUM function, so all cities are included in the remaining sum.

```
SELECT city, kind, sales,
       SUM(sales) OVER (ORDER BY city, kind
                        ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)
FROM monthly;
```

The possible results of the preceding SELECT appear in the table below.

city	kind	sales	Remaining Sum(sales)
-----	-----	-----	-----
Omaha	variety pack	25	?
Omaha	pure pork	125	25
Omaha	pure pork	25	150
Omaha	pure pork	45	175
Chicago	variety pack	55	220

Chicago	variety pack	25	275
Chicago	variety pack	45	300
Chicago	pure pork	50	345

Note that the sort order for the computation is alphabetical by city, and then by kind. The results, however, appear in the reverse order.

The sort order that you specify in the window specification defines the sort order of the rows over which the function is applied; it does not define the ordering of the results. To order the results, use an ORDER BY phrase in the SELECT statement.

For example:

```
SELECT city, kind, sales,
       SUM(sales) OVER (ORDER BY city, kind
                        ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)
FROM monthly
ORDER BY city, kind;
```

The possible results of the preceding SELECT appear in the table below:

city	kind	sales	Remaining Sum(sales)
-----	-----	-----	-----
Chicago	pure pork	50	345
Chicago	variety pack	55	265
Chicago	variety pack	25	320
Chicago	variety pack	45	220
Omaha	pure pork	25	70
Omaha	pure pork	125	95
Omaha	pure pork	45	25
Omaha	variety pack	25	?

IF you want to compute the ...	THEN use this function ...
cumulative sum	<ul style="list-style-type: none"> <li>SUM window function</li> <li>CSUM</li> </ul>
cumulative, group, or moving count	COUNT window function
group sum	SUM window function
moving average	<ul style="list-style-type: none"> <li>AVG window function</li> <li>MAVG</li> </ul>
moving difference between the current row-column value and the preceding <i>n</i> th row-column value	MDIFF
moving linear regression	MLINREG
moving sum	<ul style="list-style-type: none"> <li>SUM window function</li> </ul>

IF you want to compute the ...	THEN use this function ...
	<ul style="list-style-type: none"> <li>• MSUM</li> </ul>
quantile scores for the values in a column	QUANTILE
ordered rank of all rows in a group	<ul style="list-style-type: none"> <li>• RANK window function</li> <li>• RANK</li> </ul>
relative rank of a row in a group	PERCENT_RANK window function
sequential row number of the row within its window partition according to the window ordering of the window	ROW_NUMBER
cumulative, group, or moving maximum value	MAX window function
cumulative, group, or moving minimum value	MIN window function

## GROUP BY Clause

### GROUP BY and Window Functions

For window functions, the GROUP BY clause must include all the columns specified in the:

- Select list of the SELECT clause
- Window functions in the select list of a SELECT clause
- Window functions in the search condition of a QUALIFY clause
- The condition in the RESET WHEN clause

For example, the following SELECT statement specifies the column City in the select list and the column StoreID in the COUNT window function in the select list and QUALIFY clause. Both columns must also appear in the GROUP BY clause:

```
SELECT City, StoreID, COUNT(StoreID) OVER ()
FROM sales_tbl
GROUP BY City, StoreID
QUALIFY COUNT(StoreID) >=3;
```

For window functions, GROUP BY collapses all rows with the same value for the group-by columns into a single row.

For example, the following statement uses the GROUP BY clause to collapse all rows with the same value for City and StoreID into a single row:

```
SELECT City, StoreID, COUNT(StoreID) OVER ()
FROM sales_tbl
GROUP BY City, StoreID;
```

The results look like this:

City	StoreID	Group Count(StoreID)
-----	-----	-----
Pecos	1001	3
Pecos	1002	3
Ozona	1003	3

Without the GROUP BY, the results look like this:

City	StoreID	Group Count(StoreID)
-----	-----	-----
Pecos	1001	9
Pecos	1001	9
Pecos	1001	9
Pecos	1001	9
Pecos	1002	9
Pecos	1002	9
Pecos	1002	9
Ozona	1003	9
Ozona	1003	9

## GROUP BY and Teradata-Specific Functions

For Teradata-specific functions, GROUP BY determines the partitions over which the function executes. The clause does not collapse all rows with the same value for the group-by columns into a single row. Thus, the GROUP BY clause in these cases need only specify the partitioning column for the function.

For example, the following statement computes the running sales for each store by using the GROUP BY clause to partition the data in sales\_tbl by StoreID:

```
SELECT StoreID, Sales, CSUM(Sales, StoreID)
FROM sales_tbl
GROUP BY StoreID;
```

The results look like this:

StoreID	Sales	CSum(Sales,StoreID)
-----	-----	-----
1001	1100.00	1100.00
1001	400.00	1500.00
1001	1000.00	2500.00
1001	2000.00	4500.00
1002	500.00	500.00
1002	1500.00	2000.00

1002	2500.00	4500.00
1003	1000.00	1000.00
1003	3000.00	4000.00

## Combining Window Functions, Teradata-Specific Functions, and GROUP BY

The following table provides the semantics of the allowable combinations of window functions, Teradata-specific functions, aggregate functions, and the GROUP BY clause.

Window Function	Combination		GROUP BY Clause	Semantics
	Teradata-Specific Function	Aggregate Function		
X				A value is computed for each row.
	X			A value is computed for each row. The entire table constitutes a single group, or partition, over which the Teradata-specific function executes.
		X		One aggregate value is computed for the entire table.
X			X	GROUP BY collapses all rows with the same value for the group-by columns into a single row, and a value is computed for each resulting row.
	X		X	GROUP BY determines the partitions over which the Teradata-specific function executes. The clause does not collapse all rows with the same value for the group-by columns into a single row.
		X	X	An aggregation is performed for each group.
X	X			Teradata-specific functions do not have partitions. The whole table is one partition.
X	X		X	GROUP BY determines partitions for Teradata-specific functions. GROUP BY does not collapse all rows with the same value for the group-by columns into a single row, and does not affect window function computation.
X		X	X	GROUP BY collapses all rows with the same value for the group-by columns into a single row. For window functions, a value is computed for each resulting row; for aggregate functions, an aggregation is performed for each group.

## Possible Result Overflow with SELECT Sum

When using this function, the result can create an overflow when the data type and format are not in sync.  
For a column defined as:

```
Salary Decimal(15,2) Format '$ZZZ,ZZ9.99'
```

The following query:

```
SELECT SUM (Salary) FROM Employee;
```

causes an overflow because the decimal operand and the format are not in sync.

To avoid possible overflows, explicitly specify the format for decimal sum to specify a format large enough to accommodate the decimal sum resultant data type.

```
SELECT Sum(Salary) (format '$Z,ZZZ,ZZZ,ZZ9.99') FROM Employee;
```

## Related Topics

For more information, see:

- For descriptions of aggregate functions and arguments, see [Aggregate Functions](#).
- For more information, see “RESET WHEN Condition Rules” and “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.
- For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*.

## CSUM

### Purpose

Returns the cumulative (or running) sum of a value expression for each row in a partition, assuming the rows in the partition are sorted by the sort\_expression list.

### Type

Teradata-specific function.

### Syntax

```
— CSUM — ( — value_expression, — sort_expression — [ ASC | DESC ] ) —
```

## Syntax Elements

### value\_expression

A numeric literal or column expression for which a running sum is to be computed.

By default, CSUM uses the default data type of value\_expression. Larger numeric values are supported by casting it to a higher data type.

The expression cannot contain any ordered analytical or aggregate functions.

### sort\_expression

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

The expression cannot contain any ordered analytical or aggregate functions.

### ASC

Ascending sort order.

### DESC

Descending sort order.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using SUM Instead of CSUM

The use of CSUM is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard, and is equivalent to the ANSI-compliant SUM window function that specifies ROWS UNBOUNDED PRECEDING as its aggregation group. CSUM is retained only for backward compatibility with existing applications.

## Meaning of Cumulative Sums

CSUM accumulates a sum over an ordered set of rows, providing the current value of the SUM on each row.

## Possible Result Overflow with SELECT Sum

### Possible Result Overflow with SELECT Sum

When using this function, the result can create an overflow when the data type and format are not in sync. For a column defined as:

Salary Decimal(15,2) Format '\$ZZZ,ZZ9.99'

The following query:

```
SELECT SUM (Salary) FROM Employee;
```

causes an overflow because the decimal operand and the format are not in sync.

To avoid possible overflows, explicitly specify the format for decimal sum to specify a format large enough to accommodate the decimal sum resultant data type.

```
SELECT Sum(Salary) (format '$Z,ZZZ,ZZZ,ZZ9.99') FROM Employee;
```

## Result Type and Attributes

The data type, format, and title for CSUM are as follows:

Data Type: Same as operand x

- If operand x is character, the format is the default format for FLOAT.
- If operand x is numeric, the format is the same format as x.

## Examples

### Example

Report the daily running sales total for product code 10 for each month of 1998.

```
SELECT cmonth, CSUM(sumPrice, cdate)
FROM
  (SELECT a2.month_of_year,
   a2.calendar_date, a1.itemID, SUM(a1.price)
   FROM Sales a1, SYS_CALENDAR.Calendar a2
   WHERE a1.calendar_date=a2.calendar_date
   AND a2.calendar_date=1998
   AND a1.itemID=10
   GROUP BY a2.month_of_year, a1.calendar_date,
   a1.itemID) AS T1(cmonth, cdate, sumPrice)
GROUP BY cmonth;
```

Grouping by month allows the total to accumulate until the end of each month, when it is then set to zero for the next month. This permits the calculation of cumulative totals for each item in the same query.

## Example

Provide a running total for sales of each item in store 5 in January and generate output that is ready to export into a graphing program.

```
SELECT Item, SalesDate, CSUM(Revenue,Item,SalesDate) AS CumulativeSales
FROM
(SELECT Item, SalesDate, SUM(Sales) AS Revenue
FROM DailySales
WHERE StoreId=5 AND SalesDate BETWEEN
'1/1/1999' AND '1/31/1999'
GROUP BY Item, SalesDate) AS ItemSales
ORDER BY SalesDate;
```

The result might look something like the following table.

Item	SalesDate	CumulativeSales
InstaWoof dog food	01/01/1999	972.99
InstaWoof dog food	01/02/1999	2361.99
InstaWoof dog food	01/03/1999	5110.97
InstaWoof dog food	01/04/1999	7793.91

## CUME\_DIST

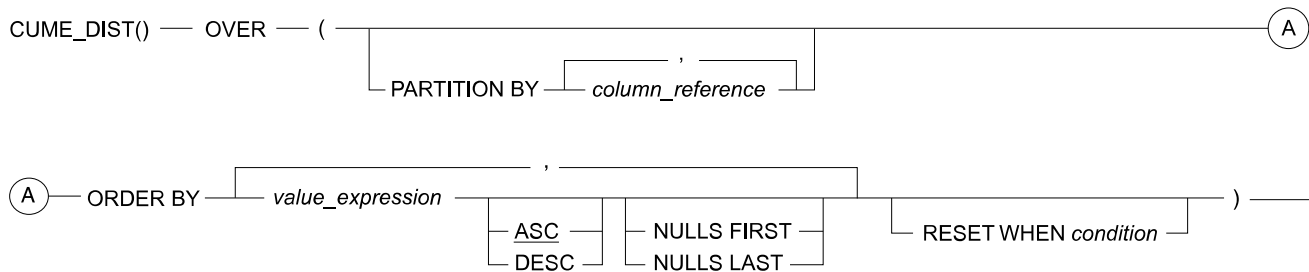
### Purpose

Calculates the cumulative distribution of a value in a group of values.

### Type

ANSI SQL:2011 window function.

## Syntax



## Syntax Elements

### OVER

Specifies how values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the `PARTITION BY` `BEGIN` and `RESET WHEN` clauses `END`, sorted according to the `ORDER BY` clause, and considered according to the aggregation group within the partition.

### PARTITION BY

The group or groups over which the function operates.

If there is no `PARTITION BY` or `RESET WHEN` clauses, then the entire result set, delivered by the `FROM` clause, constitutes a partition.

`PARTITION BY` clause is also called the window partition clause.

### ORDER BY

The order in which the values in a group or partition are sorted.

### ASC

That the results are to be ordered in ascending sort order.

If the sort field is a character string, the system orders it in ascending order according to the definition of the collation sequence for the current session.

The default order is `ASC`.

## DESC

That the results are to be ordered in descending sort order.

If the sort field is a character string, the system orders it in descending order according to the definition of the collation sequence for the current session.

Descending sort order.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

RESET WHEN is optional. If there are no RESET WHEN or PARTITION BY clauses, then the entire result set constitutes a single partition.

If there is no PARTITION BY or RESET WHEN clauses, then the entire result set, delivered by the FROM clause, constitutes a partition.

### *condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using CUME\_DIST

CUME\_DIST is similar to PERCENT\_RANK. Unlike PERCENT\_RANK, which considers the RANK value in the presence of ties, CUME\_DIST uses the highest tied rank, that is, the position of the last tied value when there are peers. CUME\_DIST is the ratio of that position in the partition (RANK-HIGH/NUM ROWS).

## Results

The range of values returned by CUME\_DIST is >0 to <=1.

## Example

The following SELECT statement:

```
SELECT lname, serviceyrs,
       CUME_DIST()OVER(ORDER BY serviceyrs)
FROM schooltbl
GROUP BY 1,2;
```

returns the cumulative distribution by service years for teachers listed in *schooltbl*.

lname	serviceyrs	CUME_DIST
Adams	10	0.333333
Peters	10	0.333333
Murray	10	0.333333
Rogers	15	0.444333
Franklin	16	0.555333
Smith	20	0.888889
Ford	20	0.888889
Derby	20	0.888889
Baker	20	1.000000

## DENSE\_RANK (ANSI)

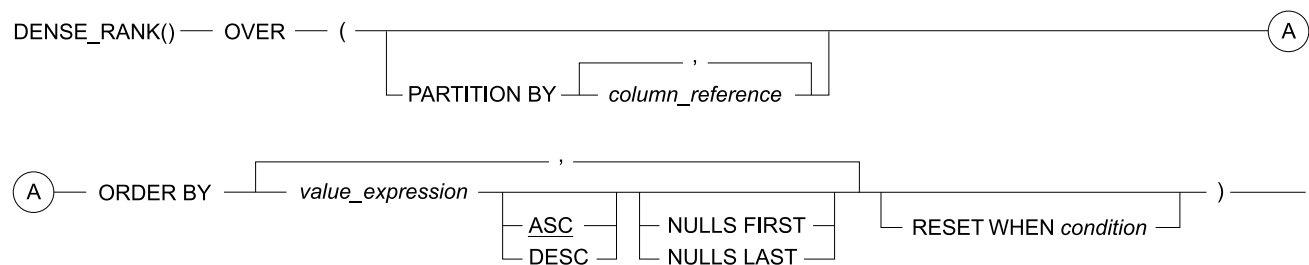
### Purpose

Returns an ordered ranking of rows based on the *value\_expression* in the ORDER BY clause.

### Type

ANSI SQL:2011 window function.

### Syntax



## Syntax Elements

### OVER

Specifies how values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the `PARTITION BY` `BEGIN` and `RESET WHEN` clauses `END`, sorted according to the `ORDER BY` clause, and considered according to the aggregation group within the partition.

### PARTITION BY

The group or groups over which the function operates.

If there is no `PARTITION BY` or `RESET WHEN` clauses, then the entire result set, delivered by the `FROM` clause, constitutes a partition.

`PARTITION BY` clause is also called the window partition clause.

### ORDER BY

The order in which the values in a group or partition are sorted.

## ASC

That the results are to be ordered in ascending sort order.

If the sort field is a character string, the system orders it in ascending order according to the definition of the collation sequence for the current session.

The default order is ASC.

## DESC

That the results are to be ordered in descending sort order.

If the sort field is a character string, the system orders it in descending order according to the definition of the collation sequence for the current session.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there are no RESET WHEN or PARTITION BY clauses, then the entire result set constitutes a single partition.

### *condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using DENSE\_RANK

The ranks are consecutive integers beginning with 1. Rows with equal values receive the same rank. Rank values are not skipped in the event of ties.

## Result Type

The result data type is INTEGER.

## Example

The following SELECT statement:

```
SELECT lname, serviceyrs,
       DENSE_RANK()OVER(ORDER BY serviceyrs)
FROM schooltbl
GROUP BY 1,2;
```

returns the ordered ranking by service years for teachers listed in *schooltbl*.

lname	serviceyrs	DENSE_RANK
Adams	10	1
Peters	10	1
Murray	10	1
Rogers	15	2
Franklin	16	3
Smith	20	4
Ford	20	4
Derby	20	4
Baker	25	5

## FIRST\_VALUE / LAST\_VALUE

### Purpose

Returns the first value or last value in an ordered set of values.

### Type

ANSI SQL:2011 window function.

### Syntax

```

— FIRST_VALUE — ( — value_expression — ) — | window |
— LAST_VALUE —
                | IGNORE — NULLS —
                | RESPECT —

```

## Syntax Elements

### *value\_expression*

A column expression.

FIRST\_VALUE and LAST\_VALUE use the default data type of *value\_expression*.

Larger numeric values are supported by casting them to a higher data type.

The expression cannot contain any ordered analytical or aggregate functions.

## IGNORE NULLS

Keyword that specifies not to return NULL.

- IGNORE NULLS (with FIRST\_VALUE) = returns the first non-null value in the set, or NULL if all values are NULL.
- If IGNORE NULLS (with LAST\_VALUE) = returns the last non-null value in the set, or NULL if all values are NULL.

## RESPECT NULLS

Optional keyword that specifies whether to return NULL.

- RESPECT NULLS (with FIRST\_VALUE) = returns the first value, whether or not it is null.
- RESPECT NULLS (with LAST\_VALUE) = returns the last value, whether or not it is null.

If all values are null, NULL is returned.

## window

A group, cumulative, or moving computation.

For Window Aggregate Function syntax, see [Window Aggregate Functions](#).

In presence of ties in the sort key of the Window Aggregate Function syntax, FIRST\_VALUE and LAST\_VALUE are non-deterministic. They return *value\_expression* from any one of the rows with tied order by value.

---

### Note:

If the ROWS phrase is omitted and there is an ORDER BY phrase, the default ROWS is UNBOUNDED PRECEDING AND CURRENT ROW.

If the ROWS phrase is omitted and there is no ORDER BY phrase, the default ROWS is UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

---

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Usage Notes

FIRST\_VALUE and LAST\_VALUE are especially valuable because they are often used as the baselines in calculations. For instance, with a partition holding sales data ordered by day, you may want to know how much the sales for each day were compared to the first sales day (FIRST\_VALUE) for the period, or you may want to know, for a set of rows in increasing sales order, what the percentage size of each sale in the region was compared to the largest sale (LAST\_VALUE) in the region.

IGNORE NULLS is particularly useful in populating an inventory table properly.

Selecting neither IGNORE NULLS or RESPECT NULLS is equivalent to selecting RESPECT NULLS.

## Example

The following example returns by start date the salary, moving average (ma), and first and last salary in the moving average group.

**Note:**

The functions are going to return the first/last value in the window. In the example, the first and last rows fall within the window. If the window were between 3 preceding and 2 preceding rows, you would see NULL for first value in the 1st two rows.

```
SELECT start_date, salary,
       AVG(salary) OVER(ORDER BY start_date
                        ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) ma,
       FIRST_VALUE(salary) OVER(ORDER BY start_date
                                ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) first,
       LAST_VALUE(salary) OVER(ORDER BY start_date
                                ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) last
FROM employee
ORDER BY start_date;
```

start_date	salary	ma	first	last
21-MAR-76	6661.78	6603.280	6661.78	6544.78
12-DEC-78	6544.78	5183.780	6661.78	2344.78
24-OCT-82	2344.78	4471.530	6661.78	2344.78
15-JAN-84	2334.78	4441.780	6661.78	4322.78
30-JUL-87	4322.980	4688.980	6544.78	7897.78
31-DEC-90	7897.78	3626.936	2344.78	1234.56
25-JUL-96	1234.56	3404.536	2334.78	1232.78
17-SEP-96	1232.78	3671.975	4322.78	1232.78

## LAG/LEAD

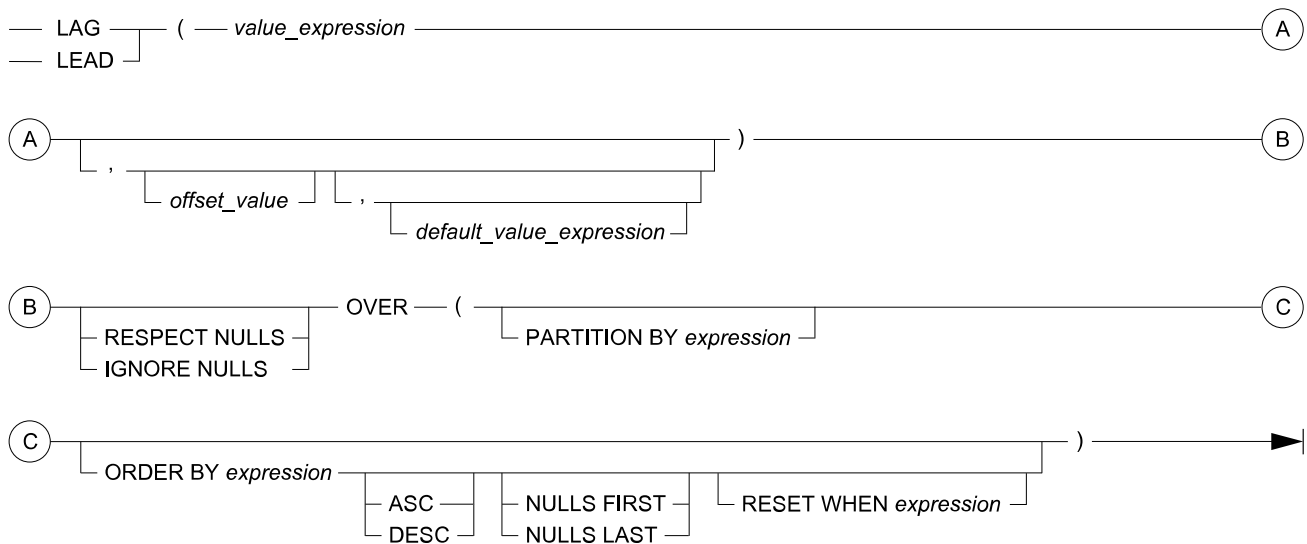
### Purpose

Ordered analytic functions calculate an aggregate or non-aggregate value on a window of rows within a group of rows. The window of rows is defined by the Window Framing clause, also called the ROWS clause. Window sizes are based on the size specified in the ROWS clause. The group of rows is defined by the PARTITION BY clause of the Window function.

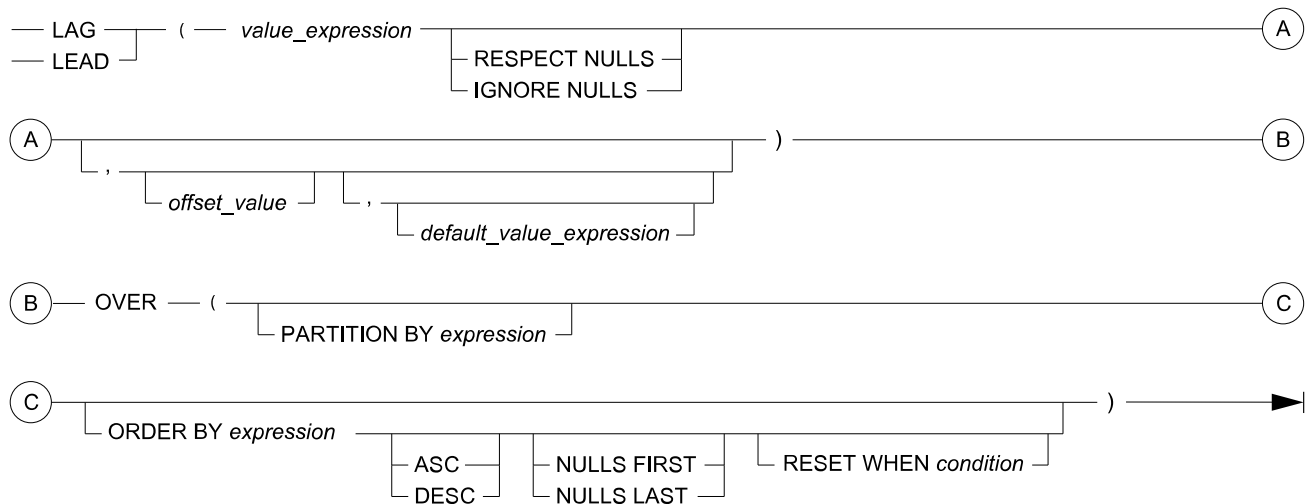
The LAG function accesses data from the row preceding the current row at a specified offset value in a window group, while the LEAD function returns data from the row following the current row. If the offset value is outside the scope of the window, the user-specified default value is returned.

The LAG and LEAD functions are used for OLAP and decision support queries.

## Syntax for LAG/LEAD ANSI Style



## Syntax for LAG/LEAD Teradata Style



## Syntax Elements

### *value\_expression*

The expression cannot contain any ordered analytical functions.

*value\_expression* is mandatory and can be any expression that returns a scalar value. It cannot be a table function.

***offset\_value***

A literal unsigned integer value between 0 and 4096. If not specified, the default value is 1.

*offset\_value* specifies the physical row position relative to the current row in a given window of rows. The row position is the row following the current row for the LEAD function, and the preceding row for the LAG function.

An *offset\_value* of 0 specifies the current row.

***default\_value\_expression***

Any expression that returns a scalar value.

If not specified, the value is assumed to be NULL.

When running in ANSI mode, the *default\_value\_expression* data type must match *value\_expression*. An error occurs if the data types do not match.

In Teradata mode, the database attempts to match the *default\_value\_expression* data type to *value\_expression* by doing a cast to *value\_expression* data type to execute the query. If there are casting rule violations, Teradata Database displays an error message.

**RESPECT NULLS**

If the preceding or following row determined by *offset\_value* is within the scope of the window group, and if the *value\_expression* evaluation returns a NULL, LAG or LEAD returns NULL. This setting indicates that the NULL value is not ignored.

If the preceding or following row is outside the scope of the window group, LAG or LEAD returns *default\_value\_expression*.

If the optional NULL clause is not specified, the default option is RESPECT NULLS.

**IGNORE NULLS**

If *value\_expression* returns a NULL value where the preceding or following row, as determined by the specified *offset\_value*, is within the scope of the window group, LAG or LEAD ignores the NULL value.

LAG or LEAD then continues searching for the non-NULL *value\_expression* in the preceding or following row, which may be far from the current row but within the scope of the window group. The search terminates at the window boundaries:

- For LAG, the search terminates at the first row of the window group.
- For LEAD, the search terminates at the last row of the window group.

At the end of the search, LAG or LEAD returns *default\_value\_expression* if no non-NULL *value\_expression* is found.

If the preceding or following row is outside the scope of the window group, LAG or LEAD returns *default\_value\_expression*.

If the optional NULL clause is not specified, the default option is RESPECT NULLS.

## OVER

Specifies how values are grouped, ordered, and considered while computing the LAG or LEAD function.

Values are grouped by the optional PARTITION BY clause and the optional RESET WHEN clause. Values are sorted according to the ORDER BY clause in a given partition of rows.

## PARTITION BY

The group or groups over which the function operates.

This is a comma-separated value expression list.

## ORDER BY

The order in which the values in a group or partition are sorted.

This is a comma-separated value expression list.

## ASC

That the results are to be ordered in ascending sort order.

If the sort field is a character string, the system orders it in ascending order according to the definition of the collation sequence for the current session.

The default order is ASC.

## DESC

Descending sort order.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there are no RESET WHEN or PARTITION BY clauses, then the entire result set constitutes a single partition.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant.

## Result Type

The data type of the LEAD or LAG function's returned values is the same as the specified value of *value\_expression*. If *default\_value\_expression* and *value\_expression* have different data types, Teradata recommends explicitly casting *default\_value\_expression* to the data type of *value\_expression*.

In ANSI mode, an error occurs if *default\_value\_expression* and *value\_expression* data types do not match.

In the Teradata Transaction (BTET) mode, if the data types do not match, the database attempts to cast the *default\_value\_expression* to the *value\_expression* data type based on the internal casting rules. If this results in casting rule violations, an error message displays.

## Usage Notes

Because the LEAD and LAG functions do not support the ROWS clause in the syntax, the window size is the same as the size of the group of rows defined by the PARTITION BY clause. If the PARTITION BY clause is absent, the entire table becomes a single group, and the size of the group of rows is the same as the total number of rows in the table.

The RESET WHEN clause, which is applicable to all window functions in Teradata Database, is extended to the LEAD and LAG functions.

The RESET WHEN clause is a Teradata Extension to ANSI. The LEAD and LAG functions support performance-driven rewrites, and support both Teradata syntax and ANSI syntax to simplify data migration from other databases.

In ANSI Transaction mode, the *value\_expression* data type must match the default value expression data type, or else an error occurs.

## Examples

### Example: LAG with IGNORE NULLS

#### ANSI style syntax:

```
SELECT empno, empname, job, sal,
       LAG(sal, 1, 0) IGNORE NULLS
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800
3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	950
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	2450
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0
5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	1250
8	MATTHEW	SALESMAN	1600	1500

#### Teradata style syntax:

```
SELECT empno, empname, job, sal,
       LAG(sal IGNORE NULLS, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800

3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	950
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	2450
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0
5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	1250
8	MATTHEW	SALESMAN	1600	1500

## Example: LAG with RESPECT NULLS

### ANSI style syntax:

```
SELECT empno, empname, job, sal,
       LAG(sal, 1, 0) RESPECT NULLS
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
-----				
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	?
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800
3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	?
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	?
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0
5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	?
8	MATTHEW	SALESMAN	1600	1500

### Teradata style syntax:

```
SELECT empno, empname, job, sal,
       LAG(sal RESPECT NULLS, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
```

```
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	?
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800
3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	?
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	?
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0
5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	?
8	MATTHEW	SALESMAN	1600	1500

## Example: LAG with RESPECT NULLS without Explicitly Specifying RESPECT NULLS

### ANSI style syntax:

```
SELECT empno, empname, job, sal,
       LAG (sal, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	?
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800
3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	?
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	?
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0

5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	?
8	MATTHEW	SALESMAN	1600	1500

**Teradata style syntax:**

```
SELECT empno, empname, job, sal,
       LAG (sal, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_prev
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_PREV
-----				
12	PAUL	ANALYST	?	0
13	GRACE	ANALYST	3000	?
1	JOHN	CLERK	800	0
2	ERIC	CLERK	950	800
3	KURT	CLERK	?	950
6	JULIE	CLERK	1300	?
9	NICHOLAS	MANAGER	2450	0
10	NOVAK	MANAGER	?	2450
11	ROGER	MANAGER	2850	?
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	0
5	LYNN	SALESMAN	?	1250
7	TERESA	SALESMAN	1500	?
8	MATTHEW	SALESMAN	1600	1500

**Example: LEAD with RESPECT NULLS****ANSI style syntax:**

```
SELECT empno, empname, job, sal,
       LEAD(sal, 1, 0) RESPECT NULLS
       OVER (PARTITION BY job ORDER BY empno) AS sal_next
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
-----				
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950

2	ERIC	CLERK	950	?
3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	?
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	?
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

**Teradata style syntax:**

```
SELECT empno, empname, job, sal,
       LEAD(sal RESPECT NULLS, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_next
FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
-----				
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950
2	ERIC	CLERK	950	?
3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	?
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	?
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

**Example: LEAD with IGNORE NULLS****ANSI style syntax:**

```
SELECT empno, empname, job, sal,
       LEAD(sal, 1, 0) IGNORE NULLS
       OVER (PARTITION BY job ORDER BY empno) AS sal_next
```

```
FROM emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950
2	ERIC	CLERK	950	1300
3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	2850
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	1500
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

### Teradata style syntax:

```
SELECT empno, empname, job, sal,
       LEAD(sal IGNORE NULLS, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_next
FROM emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950
2	ERIC	CLERK	950	1300
3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	2850
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	1500
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

## Example: LEAD with RESPECT NULLS without Explicitly Specifying RESPECT NULLS

### ANSI style syntax:

```
SELECT empno, empname, job, sal,
       LEAD (sal, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_next

FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950
2	ERIC	CLERK	950	?
3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	?
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	?
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

### Teradata style syntax:

```
SELECT empno, empname, job, sal,
       LEAD (sal, 1, 0)
       OVER (PARTITION BY job ORDER BY empno) AS sal_next

FROM   emp
ORDER BY job, empno;
```

EMPNO	EMPNAME	JOB	SAL	SAL_NEXT
12	PAUL	ANALYST	?	3000
13	GRACE	ANALYST	3000	0
1	JOHN	CLERK	800	950
2	ERIC	CLERK	950	?

3	KURT	CLERK	?	1300
6	JULIE	CLERK	1300	0
9	NICHOLAS	MANAGER	2450	?
10	NOVAK	MANAGER	?	2850
11	ROGER	MANAGER	2850	0
14	RICH	PRESIDENT	5000	0
4	KENT	SALESMAN	1250	?
5	LYNN	SALESMAN	?	1500
7	TERESA	SALESMAN	1500	1600
8	MATTHEW	SALESMAN	1600	0

## MAVG

### Purpose

Computes the moving average of a value expression for each row in a partition using the specified value expression for the current row and the preceding *width*-1 rows.

### Type

Teradata-specific function.

### Syntax

```
MAVG ( value_expression, width, sort_expression [ASC|DESC] )
```

## Syntax Elements

### *value\_expression*

The expression cannot contain any ordered analytical or aggregate functions.

### *width*

The number of previous rows to be used in the computation.

The value is always a positive integer literal.

The maximum is 4096.

### *sort\_expression*

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

For example, MAVG(Sale, 6, Region ASC, Store DESC), where Sale is the *value\_expression*, 6 is the *width*, and Region ASC, Store DESC is the *sort\_expression* list.

The expression cannot contain any ordered analytical or aggregate functions.

## ASC

That the results are to be ordered in ascending sort order.

If the sort field is a character string, the system orders it in ascending order according to the definition of the collation sequence for the current session.

The default order is ASC.

## DESC

That the results are to be ordered in descending sort order.

If the sort field is a character string, the system orders it in descending order according to the definition of the collation sequence for the current session.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using AVG Instead of MAVG

The use of MAVG is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard, and is equivalent to the ANSI-compliant AVG window function that specifies ROWS *value* PRECEDING as its aggregation group. MAVG is retained only for backward compatibility with existing applications.

## Result Type and Attributes

The data type, format, and title for MAVG are as follows:

Data Type: Same as operand x

- If operand x is character, the format is the default format for FLOAT.
- If operand x is numeric, date, or interval, the format is the same format as x.

## Problems With Missing Data

Ensure that data you analyze using MAVG has no missing data points. Computing a moving average over data with missing points produces unexpected and incorrect results because the computation considers  $n$  physical rows of data rather than  $n$  logical data points.

## Computing the Moving Average When Number of Rows < width

For the (possibly grouped) resulting relation, the moving average considering *width* rows is computed where the rows are sorted by the *sort\_expression* list.

When there are fewer than *width* rows, the average is computed using the current row and all preceding rows.

## Examples

### Example

Compute the 7-day moving average of sales for product code 10 for each day in the month of October, 1996.

```
SELECT cdate, itemID, MAVG(sumPrice, 7, date)
FROM (SELECT a1.calendar_date, a1.itemID,
SUM(a1.price)
FROM Sales a1
WHERE a1.itemID=10 AND a1.calendar_date
BETWEEN 96-10-01 AND 96-10-31
GROUP BY a1.calendar_date, a1.itemID) AS T1(cdate,
itemID, sumPrice);
```

### Example

The following example calculates the 50-day moving average of the closing price of the stock for Zemlinsky Bros. Corporation. The ticker name for the company is ZBC.

```
SELECT MarketDay, ClosingPrice,
MAVG(ClosingPrice,50, MarketDay) AS ZBCAverage
FROM MarketDailyClosing
WHERE Ticker = 'ZBC'
ORDER BY MarketDay;
```

The results for the query might look something like the following table.

MarketDay	ClosingPrice	ZBCAverage
12/27/1999	89 1/16	85 1/2
12/28/1999	91 1/8	86 1/16
12/29/1999	92 3/4	86 1/2
12/30/1999	94 1/2	87

## MDIFF

### Purpose

Returns the moving difference between the specified value expression for the current row and the preceding *width* rows for each row in the partition.

### Type

Teradata-specific function.

### Syntax

```

— MDIFF — ( — value_expression, — width, — sort_expression — )
                                     [ ASC ]
                                     [ DESC ]

```

## Syntax Elements

### *value\_expression*

A numeric column or literal expression for which a moving difference is to be computed.

The expression cannot contain any ordered analytical or aggregate functions.

### *width*

The number of previous rows to be used in the computation.

The value is always a positive integer literal.

The maximum is 4096.

### *sort\_expression*

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

### **ASC**

Ascending sort order.

### **DESC**

Descending sort order.

## **ANSI Compliance**

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## **Meaning of Moving Difference**

A common business metric is to compare activity for some variable in a current time period to the activity for the same variable in another time period a fixed distance in the past. For example, you might want to compare current sales volume against sales volume for preceding quarters. This is a moving difference calculation where *value\_expression* would be the quarterly sales volume, width is 4, and *sort\_expression* might be the *quarter\_of\_calendar* column from the *SYS\_CALENDAR.Calendar* system view.

## **Using SUM Instead of MDIFF**

The use of MDIFF is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard, and is retained only for backward compatibility with existing applications. MDIFF(x, w, y) is equivalent to:

```
x - SUM(x) OVER (ORDER BY y
                 ROWS BETWEEN w PRECEDING AND w PRECEDING)
```

## **Result Type and Attributes**

The data type, format, and title for MDIFF are as follows:

- If operand x is character, the data type is the same as x and the format is the default format for FLOAT.
- If operand x is numeric, the data type is the same as x and the format is the same format as x.
- If operand is date, the data type is INTEGER and the format is the default format for INTEGER.

## Problems With Missing Data

Ensure that rows you analyze using MDIFF have no missing data points. Computing a moving difference over data with missing points produces unexpected and incorrect results because the computation considers  $n$  physical rows of data rather than  $n$  logical data points.

## Computing the Moving Difference When No Preceding Row Exists

When the number of preceding rows to use in a moving difference computation is fewer than the specified width, the result is null.

## Examples

### Example

Display the difference between each quarter and the same quarter sales for last year for product code 10.

```
SELECT year_of_calendar, quarter_of_calendar,
MDIFF(sumPrice, 4, year_of_calendar, quarter_of_calendar)
FROM (SELECT a2.year_of_calendar,
a2.quarter_of_calendar, SUM(a2.Price) AS sumPrice
FROM Sales a1, SYS_CALENDAR.Calendar a2
WHERE a1.itemID=10 and a1.calendar_date=a2.calendar_date
GROUP BY a2.year_of_calendar, a2.quarter_of_calendar) AS T1
ORDER BY year_of_calendar, quarter_of_year;
```

### Example

The following example computes the changing market volume week over week for the stock of company Horatio Parker Imports. The ticker name for the company is HPI.

```
SELECT MarketWeek, WeekVolume,
MDIFF(WeekVolume,1,MarketWeek) AS HPIVolumeDiff
FROM
(SELECT MarketWeek, SUM(Volume) AS WeekVolume
FROM MarketDailyClosing
WHERE Ticker = 'HPI'
GROUP BY MarketWeek)
ORDER BY MarketWeek;
```

The result might look like the following table. Note that the first row is null for column HPIVolume Diff, indicating no previous row from which to compute a difference.

MarketWeek	WeekVolume	HPIVolumeDiff
11/29/1999	9817671	?
12/06/1999	9945671	128000
12/13/1999	10099459	153788
12/20/1999	10490732	391273
12/27/1999	11045331	554599

## Related Topics

For more information, see:

- For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on the SUM window function, see [Window Aggregate Functions](#).

## MEDIAN

### Purpose

For numeric values, returns the middle value or an interpolated value that would be the middle value after the values are sorted. Nulls are ignored in the calculation.

### Type

MEDIAN is an aggregate function.

### Syntax

MEDIAN — (value\_expression) —

## Syntax Elements

*value\_expression*

A single expression that must be a numeric or interval data type.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Result

The function returns the same data type as the data type of the argument.

## Example

MEDIAN, an inverse distribution function that assumes a continuous distribution model, is a specific case of PERCENTILE\_CONT where the percentile value is 0.5.

```
MEDIAN (value_expression)
```

is same as:

```
PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY value_expression)
```

## Related Topics

For more information, see:

- See [PERCENTILE\\_CONT / PERCENTILE\\_DISC](#).

## MLINREG

### Purpose

Returns a predicted value for an expression based on a least squares moving linear regression of the previous *width* -1 (based on *sort\_expression*) column values.

### Type

Teradata-specific function.

### Syntax

```
— MLINREG — ( — value_expression, — width, — sort_expression — 

|      |
|------|
| ASC  |
| DESC |

 ) —
```

## Syntax Elements

### *value\_expression*

The expression cannot contain any ordered analytical or aggregate functions.

### *width*

The number of previous rows to be used in the computation.

The value is always a positive integer literal.

The maximum is 4096.

### *sort\_expression*

A column expression that defines the independent variable for calculating the linear regression.

For example, `MLINREG(Sales, 6, Fiscal_Year_Month ASC)`, where `Sales` is the *value\_expression*, `6` is the *width*, and `Fiscal_Year_Month ASC` is the *sort\_expression*.

The data type of the column reference must be numeric or a data type that Teradata Database can successfully convert implicitly to numeric.

## ASC

Ascending sort order.

## DESC

Descending sort order.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using ANSI-Compliant Window Functions Instead of MLINREG

Using ANSI-compliant window functions instead of `MLINREG` is strongly encouraged. `MLINREG` is a Teradata extension to the ANSI SQL:2011 standard, and is retained only for backward compatibility with existing applications.

## Result Type and Attributes

The data type, format, and title for MLINREG are as follows:

Data Type: Same as operand x

- If operand x is character, the format is the default format for FLOAT.
- If operand x is numeric, date, or interval, the format is the same format as x.

## Default Independent Variable

MLINREG assumes that the independent variable is described by *sort\_expression*.

## Computing MLINREG When Preceding Rows < width - 1

When there are fewer than *width* - 1 preceding rows, MLINREG computes the regression using all the preceding rows.

## MLINREG Report Structure

All rows in the results table except the first two, which are always null, display the predicted value.

## Example

Consider the *itemID*, *smonth*, and *sales* columns from *sales\_table*:

```
SELECT itemID, smonth, sales
FROM fiscal_year_sales_table
ORDER BY itemID, smonth;
itemID  smonth  sales
-----  -
A        1      100
A        2      110
A        3      120
A        4      130
A        5      140
A        6      150
A        7      170
A        8      190
A        9      210
A       10      230
A       11      250
A       12      ?
B        1      20
```

```
B          2      30
...
```

Assume that the NULL in the *sales* column is because in this example the month of December (month 12) is a future date and the value is unknown.

The following statement uses MLINREG to display the expected sales using past trends for each month for each product using the sales data for the previous six months.

```
SELECT itemID, smonth, sales, MLINREG(sales,7,smonth)
FROM fiscal_year_sales_table;
GROUP BY itemID;
```

itemID	smonth	sales	MLinReg(sales,7,smonth)
A	1	100	?
A	2	110	?
A	3	120	120
A	4	130	130
A	5	140	140
A	6	150	150
A	7	170	160
A	8	190	177
A	9	210	198
A	10	230	222
A	11	250	247
A	12	?	270
B	1	20	?
B	2	30	?
...			

## Related Topics

For information on the default format of data types and an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## MSUM

### Purpose

Computes the moving sum specified by a value expression for the current row and the preceding *n*-1 rows. This function is very similar to the MAVG function.

### Type

Teradata-specific function.

## Syntax

```

— MSUM — ( — value_expression, — width, — sort_expression — )
                                     |
                                     |— ASC —
                                     |— DESC —

```

## Syntax Elements

### *value\_expression*

The expression cannot contain any ordered analytical or aggregate functions.

### *width*

The number of previous rows to be used in the computation.

The value is always a positive integer literal.

The maximum is 4096.

### *sort\_expression*

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

## ASC

Ascending sort order.

## DESC

Descending sort order.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using SUM Instead of MSUM

The use of MSUM is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard, and is equivalent to the ANSI-compliant SUM window function. MSUM is retained only for backward compatibility with existing applications.

## Result Type and Attributes

The data type, format, and title for MSUM are as follows:

Data Type: Same as operand *x*

- If operand *x* is character, the format is the default format for FLOAT.
- If operand *x* is numeric, the format is the same format as *x*.

## Problems With Missing Data

Ensure that data you analyze using MSUM has no missing data points. Computing a moving average over data with missing points produces unexpected and incorrect results because the computation considers *n* physical rows of data rather than *n* logical data points.

## Computing MSUM When Number of Rows < width

For data having fewer than *width* rows, MSUM computes the sum using all the preceding rows. MSUM returns the current sum rather than nulls when the number of rows in the sample is fewer than *width*.

## Possible Result Overflow with SELECT Sum

When using this function, the result can create an overflow when the data type and format are not in sync. For a column defined as:

```
Salary Decimal(15,2) Format '$ZZZ,ZZ9.99'
```

The following query:

```
SELECT SUM (Salary) FROM Employee;
```

causes an overflow because the decimal operand and the format are not in sync.

To avoid possible overflows, explicitly specify the format for decimal sum to specify a format large enough to accommodate the decimal sum resultant data type.

```
SELECT Sum(Salary) (format '$Z,ZZZ,ZZZ,ZZ9.99') FROM Employee;
```

# PERCENT\_RANK

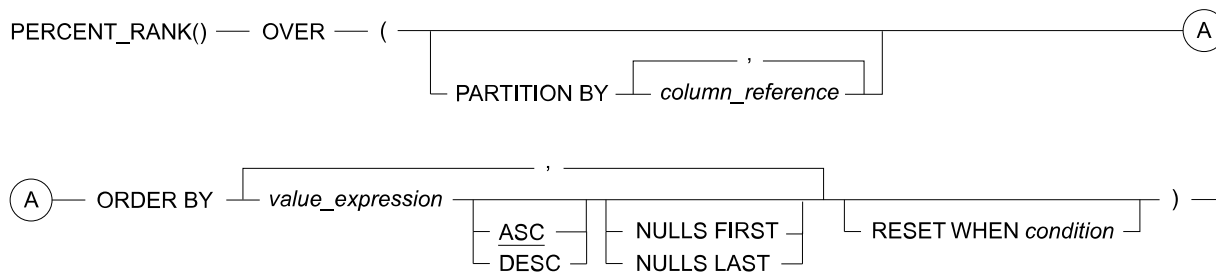
## Purpose

Returns the relative rank of rows for a *value\_expression*.

## Type

ANSI SQL:2011 window function.

## Syntax



## Syntax Elements

### OVER

Specifies how values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the `PARTITION BY` `BEGIN` and `RESET WHEN` clauses `END`, sorted according to the `ORDER BY` clause, and considered according to the aggregation group within the partition.

### PARTITION BY

The group or groups over which the function operates.

If there is no `PARTITION BY` or `RESET WHEN` clauses, then the entire result set, delivered by the `FROM` clause, constitutes a partition.

`PARTITION BY` clause is also called the window partition clause.

### ORDER BY

The order in which the values in a group or partition are sorted.

**ASC**

Ascending sort order.

**DESC**

Descending sort order.

**NULLS FIRST**

NULL results are to be listed first.

**NULLS LAST**

NULL results are to be listed last.

**RESET WHEN**

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there is no PARTITION BY or RESET WHEN clauses, then the entire result set, delivered by the FROM clause, constitutes a partition.

*condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

**ANSI Compliance**

This is ANSI SQL:2011 compliant.

The RESET WHEN clause is a Teradata extension to the ANSI SQL standard.

## Computation

The assigned rank of a row is defined as 1 (one) plus the number of rows that precede the row and are not peers of it.

PERCENT\_RANK is expressed as an approximate numeric ratio between 0.0 and 1.0.

PERCENT_RANK has this value ...	FOR the result row assigned this rank ...
0.0	1.
1.0	highest in the result.

## Result Type and Attributes

For PERCENT\_RANK() OVER (PARTITION BY *x* ORDER BY *y direction* ), the data type, format, and title are as follows:

Data Type	Format	Title
REAL	the default format for DECIMAL(7,6).	Percent_Rank(y direction)

## Examples

### Example: Relative Rank

Determine the relative rank, called the percent\_rank, of Christmas sales.

The following query:

```
SELECT sales_amt,
       PERCENT_RANK() OVER (ORDER BY sales_amt)
FROM xsales;
```

might return the following results. Note that the relative rank is returned in ascending order, the default when no sort order is specified and that the currency is not reported explicitly.

sales_amt	Percent_Rank
100.00	0.000000
120.00	0.125000
130.00	0.250000

sales_amt	Percent_Rank
140.00	0.375000
143.00	0.500000
147.00	0.625000
150.00	0.750000
155.00	0.875000
160.00	1.000000

### Example: Rank and Relative Rank

Determine the rank and the relative rank of Christmas sales.

```
SELECT sales_amt,
       RANK() OVER (ORDER BY sales_amt),
       PERCENT_RANK () OVER (ORDER BY sales_amt)
FROM xsales;
```

sales_amt	Rank	Percent_Rank
100.00	1	0.000000
120.00	2	0.125000
130.00	3	0.250000
140.00	4	0.375000
143.00	5	0.500000
147.00	6	0.625000
150.00	7	0.750000
155.00	8	0.875000
160.00	9	1.000000

### Example: PERCENT\_RANK and CUM\_DIST

The following SQL statement illustrates the difference between PERCENT\_RANK and cumulative distribution.

```
SELECT sales_amt,  
       PERCENT_RANK() OVER (ORDER BY sales_amt),  
       CUME_DIST() OVER (ORDER BY sales_amt)  
FROM xsales;
```

sales_amt	PERCENT_Rank	CUME_DIST
100.	.000000	0.125000
120.	.142857	0.250000
130	.285714	.375000
140.	.428571	.500000
147.	.571429	.625000
150.	.714286	.750000
155.	.857143	.875000
160.	1.000000	1.000000

# PERCENTILE\_CONT / PERCENTILE\_DISC

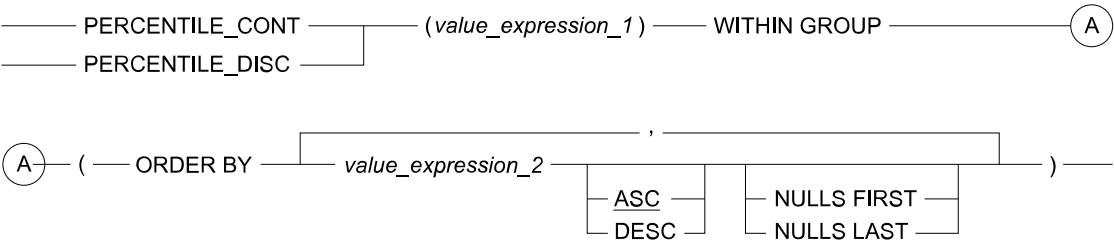
## Purpose

Returns an interpolated value that falls within its *value\_expression* with respect to its sort specification.

## Type

PERCENTILE\_CONT and PERCENTILE\_DISC are aggregate functions.

## Syntax



## Syntax Elements

*value\_expression\_1*

A numeric value between 0 and 1 inclusive.

### **WITHIN GROUP**

The order in which the values in a group or partition are sorted.

### **ORDER BY**

The order in which the values in a group or partition are sorted.

*value\_expression\_2*

A single expression that must be a numeric value.

### **ASC**

Ascending sort order.

### **DESC**

Descending sort order.

### **NULLS FIRST**

NULL results are to be listed first.

### **NULLS LAST**

NULL results are to be listed last.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Result

The function returns the same data type as the data type of the argument.

Nulls are ignored in the calculation.

## Usage Notes

Both functions are inverse distribution functions that assume a continuous distribution.

- PERCENTILE\_CONT returns a computed result after doing linear interpolation.
- PERCENTILE\_DISC simply returns a value from the set of values.

## Example

Using this table:

Area	Address	Price
Downtown	72 Easy Street	509000
Downtown	29 Right Way	402000
Downtown	45 Diamond Lane	203000
Downtown	76 Blind Alley	201000
Downtown	15 Tern Pike	199000
Downtown	444 Kanga Road	102000
Uptown	15 Peak Street	456000
Uptown	27 Primrose Path	349000
Uptown	44 Shady Lane	341000
Uptown	34 Design Road	244000
Uptown	2331 Highway 64	244000
Uptown	77 Sunset Strip	102000

the following SQL statement returns a computed result after doing linear interpolation, as shown in the table immediately below.

```
SELECT area,
       AVG(price),
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY price),
       PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY price)
```

```
FROM market
GROUP BY area;
```

Area	Average Price	PERC_DISC	PERC_CONT
Downtown	269333	201000	202000
Uptown	289333	244000	292500

## QUANTILE

### Purpose

Computes the quantile scores for the values in a group.

### Type

Teradata-specific function.

### Syntax

```
— QUANTILE — ( — quantile_literal, — sort_expression — ) —
                                     |
                                     | ASC
                                     | DESC
```

## Syntax Elements

### *quantile\_literal*

A positive integer literal used to define the number of quantile partitions to be used.

### *sort\_expression*

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

### ASC

Ascending sort order.

## DESC

Descending sort order.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Definition

A quantile is a generic interval of user-defined width. For example, percentiles divide data among 100 evenly spaced intervals, deciles among 10 evenly spaced intervals, quartiles among 4, and so on. A quantile score indicates the fraction of rows having a *sort\_expression* value lower than the current value. For example, a percentile score of 98 means that 98 percent of the rows in the list have a *sort\_expression* value lower than the current value.

## Using ANSI Window Functions Instead of QUANTILE

The use of QUANTILE is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard and is retained only for backward compatibility with existing applications.

To compute QUANTILE(q, s) using ANSI window functions, use the following:

```
(RANK() OVER (ORDER BY s) - 1) * q / COUNT(*) OVER()
```

## QUANTILE Report

For each row in the group, QUANTILE returns an integer value that represents the quantile of the *sort\_expression* value for that row relative to the *sort\_expression* value for all the rows in the group.

## Quantile Value Range

Quantile values range from 0 through (Q-1), where Q is the number of quantile partitions specified by *quantile\_literal*.

## Result Type and Attributes

The data type, format, and title for QUANTILE(Q, list) are as follows.

Data Type	Format	Title
INTEGER	the default format for the INTEGER data type	Quantile(Q, list)

## Examples

### Example

Display each item and its total sales in the ninth (top) decile according to the total sales.

```
SELECT itemID, sumPrice
FROM (SELECT a1.itemID, SUM(price)
FROM Sales a1
GROUP BY a1.itemID) AS T1(itemID, sumPrice)
QUALIFY QUANTILE(10,sumPrice)=9;
```

### Example

The following example groups all items into deciles by profitability.

```
SELECT Item, Profit, QUANTILE(10, Profit) AS Decile
FROM
  (SELECT Item, Sum(Sales) – (Count(Sales) * ItemCost) AS Profit
  FROM DailySales, Items
  WHERE DailySales.Item = Items.Item
  GROUP BY Item) AS Item;
```

The result might look like the following table.

Item	Profit	Decile
High Tops	97112	9
Low Tops	74699	7
Running	69712	6
Casual	28912	3
Xtrain	100129	9

### Example

Because QUANTILE uses equal-width histograms to partition the specified data, it does not partition the data equally using equal-height histograms. In other words, do not expect equal row counts per specified quantile. Expect empty quantile histograms when, for example, duplicate values for *sort\_expression* are found in the data.

For example, consider the following simple SELECT statement.

```
SELECT itemNo, quantity, QUANTILE(10,quantity) FROM inventory;
```

The report might look like this.

itemNo	quantity	Quantile(10, quantity)
13	1	0
9	1	0
7	1	0
2	1	0
5	1	0
3	1	0
1	1	0
6	1	0
4	1	0
10	1	0
8	1	0
11	1	0
12	9	9

Because the quantile sort is on quantity, and there are only two quantity scores in the inventory table, there are no scores in the report for deciles 1 through 8.

## Related Topics

For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## RANK (ANSI)

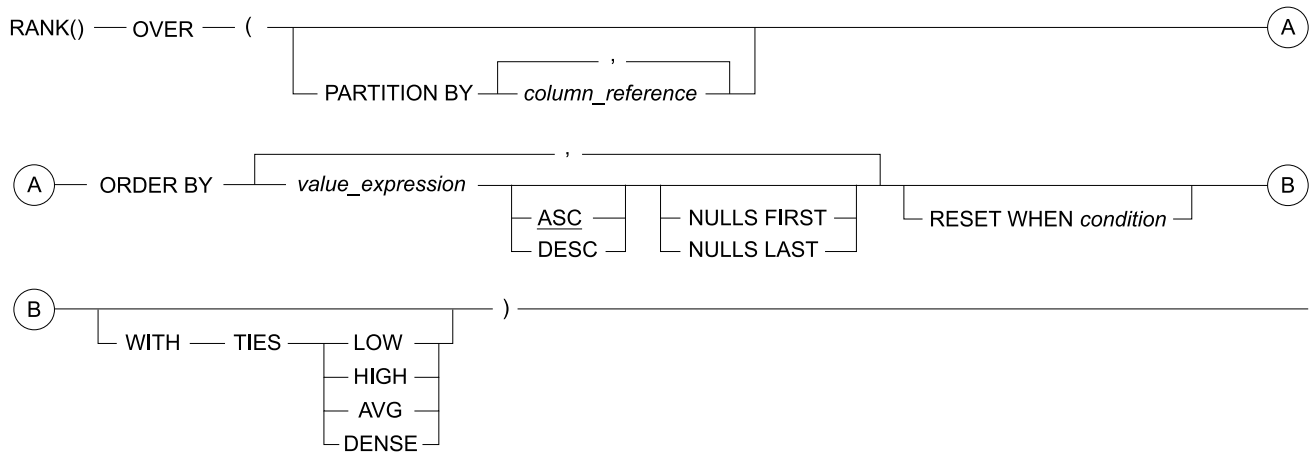
### Purpose

Returns an ordered ranking of rows based on the *value\_expression* in the ORDER BY clause.

To use this function with time series data, see *Teradata Vantage™ Time Series Tables and Operations*, B035-1208.

**Type**

ANSI SQL:2011 window function.

**Syntax****Syntax Elements****OVER**

Specifies how values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the `PARTITION BY` `BEGIN` and `RESET WHEN` clauses `END`, sorted according to the `ORDER BY` clause, and considered according to the aggregation group within the partition.

**PARTITION BY**

The group or groups over which the function operates.

If there is no `PARTITION BY` or `RESET WHEN` clauses, then the entire result set, delivered by the `FROM` clause, constitutes a partition.

`PARTITION BY` clause is also called the window partition clause.

**ORDER BY**

The order in which the values in a group or partition are sorted.

## ASC

Ascending sort order.

## DESC

Descending sort order.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there is no PARTITION BY or RESET WHEN clauses, then the entire result set, delivered by the FROM clause, constitutes a partition.

### *condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

## TIES LOW

Specifies that all ties get the lowest rank.

Returns an Integer data type.

## TIES HIGH

Specifies that all ties get the highest rank.

Returns an Integer data type.

## TIES AVG

Specifies that all ties get the average rank.

Returns a Decimal data type.

## TIES DENSE

Specifies that all ties are ranked as DENSE\_RANK ranks them.

Returns an Integer data type.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant, but includes non-ANSI Teradata Database extensions.

## Meaning of Rank

RANK returns an ordered ranking of rows based on the *value\_expression* in the ORDER BY clause. All rows having the same *value\_expression* value are assigned the same rank.

If  $n$  rows have the same *value\_expression* values, then they are assigned the same rank, call it rank  $r$ . The next distinct value receives rank  $r + n$ . And so on.

Less formally, RANK sorts a result set and identifies the numeric rank of each row in the result. RANK returns an integer that represents the rank of each row in the result.

## Result Type and Attributes

For RANK() OVER (PARTITION BY  $x$  ORDER BY  $y$  *direction*), the data type, format, and title are as follows.

Data Type	Format	Title
INTEGER	the default format for the INTEGER data type	Rank( $y$ direction)

## Examples

### Example: Ranking Salespeople Based on Sales

This example ranks salespersons by sales region based on their sales.

```
SELECT sales_person, sales_region, sales_amount,
       RANK() OVER (PARTITION BY sales_region ORDER BY sales_amount DESC)
FROM sales_table;
```

sales_person	sales_region	sales_amount	Rank(sales_amount)
Garabaldi	East	100	1
Baker	East	99	2
Fine	East	89	3
Adams	East	75	4
Edwards	West	100	1
Connors	West	99	2
Davis	West	99	2

The rank column in the preceding table lists salespersons in declining sales order according to the column specified in the PARTITION BY clause (sales\_region) and that the rank of their sales (sales\_amount) is reset when the sales\_region changes.

### Example: Finding Differences Between RANK(ANSI) and DENSE\_RANK(ANSI)

The following SQL statement illustrates the difference between RANK(ANSI) and DENSE\_RANK(ANSI), returning the RANK and DENSE\_RANK for sales\_person by sales\_region and sales\_amount.

```
SELECT sales_person, sales_region, sales_amount,
       RANK() OVER
         (PARTITION BY sales_region ORDER BY sales_amount DESC) as "Rank",
       DENSE_RANK() OVER
         (PARTITION BY sales_region ORDER BY sales_amount DESC) as "DenseRank"
FROM sales_table;
```

sales_person	sales_region	sales_amount	Rank	DenseRank
Garabaldi	East	100	1	1

sales_person	sales_region	sales_amount	Rank	DenseRank
Baker	East	100	1	1
Fine	East	89	3	2
Adams	East	75	4	3
Edwards	West	100	1	1
Connors	West	99	2	2
Davis	West	99	2	2
Russell	West	50	4	3

## Related Topics

For more information, see:

- For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.
- For an explanation of the formatting characters in the format, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.

## RANK (Teradata)

## Purpose

Returns the rank (1 ... *n*) of all the rows in the group by the value of *sort\_expression* list, with the same *sort\_expression* values receiving the same rank.

## Type

Teradata-specific function.

## Syntax

— RANK — ( — *sort\_expression* — ( — ASC —  
DESC — ) — ) —

## Syntax Elements

### *sort\_expression*

A literal or column expression or comma-separated list of literal or column expressions to be used to sort the values.

The expression cannot contain any ordered analytical or aggregate functions.

### ASC

Ascending sort order.

### DESC

Descending sort order.

## ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

## Using ANSI RANK Instead of Teradata RANK

The use of Teradata RANK is strongly discouraged. It is a Teradata extension to the ANSI SQL:2011 standard, and is equivalent to the ANSI-compliant RANK window function. Teradata RANK is retained only for backward compatibility with existing applications.

## Meaning of Rank

A rank  $r$  implies the existence of exactly  $r-1$  rows with *sort\_expression* value preceding it. All rows having the same *sort\_expression* value are assigned the same rank.

For example, if  $n$  rows have the same *sort\_expression* values, then they are assigned the same rank, call it rank  $r$ . The next distinct value receives rank  $r + n$ .

Less formally, RANK sorts a result set and identifies the numeric rank of each row in the result. The only argument for RANK is the sort column or columns, and the function returns an integer that represents the rank of each row in the result.

## Computing Top and Bottom Values

You can use RANK to compute top and bottom values as shown in the following examples.

Top( $n$ , column) is computed as QUALIFY RANK(column DESC)  $\leq n$ .

Bottom( $n$ , column) is computed as `QUALIFY RANK(column ASC) <=n`.

## Result Type and Attributes

The data type, format, and title for `RANK(x)` are as follows.

Data Type	Format	Title
INTEGER	the default format for the INTEGER data type	Rank(x)

## Examples

### Example

Display each item, its total sales, and its sales rank for the top 100 selling items.

```
SELECT itemID, sumPrice, RANK(sumPrice)
FROM
  (SELECT a1.itemID, SUM(a1.Price)
   FROM Sales a1
   GROUP BY a1.itemID AS T1(itemID, sumPrice)
   QUALIFY RANK(sumPrice) <=100;
```

### Example

Sort employees alphabetically and identify their level of seniority in the company.

```
SELECT EmployeeName, (HireDate - CURRENT_DATE) AS ServiceDays,
RANK(ServiceDays) AS Seniority
FROM Employee
ORDER BY EmployeeName;
```

The result might look like the following table.

EmployeeName	Service Days	Seniority
Ferneyhough	9931	2
Lucier	9409	4
Revueltas	9408	5
Ung	9931	2
Wagner	10248	1

## Example

Sort items by category and report them in order of descending revenue rank.

```
SELECT Category, Item, Revenue, RANK(Revenue) AS ItemRank
FROM ItemCategory,
     (SELECT Item, SUM(sales) AS Revenue
      FROM DailySales
      GROUP BY Item) AS ItemSales
WHERE ItemCategory.Item = ItemSales.Item
ORDER BY Category, ItemRank DESC;
```

The result might look like the following table.

Category	Item	Revenue	ItemRank
Hot Cereal	Regular Oatmeal	39112.00	4
Hot Cereal	Instant Oatmeal	44918.00	3
Hot Cereal	Regular COW	59813.00	2
Hot Cereal	Instant COW	75411.00	1

## Related Topics

For more information, see:

- For information on the default format of data types, see “Data Type Formats and Format Phrases” in *Teradata Vantage™ Data Types and Literals*, B035-1143.
- For more information on the RANK window function, see [RANK \(ANSI\)](#).

## ROW\_NUMBER

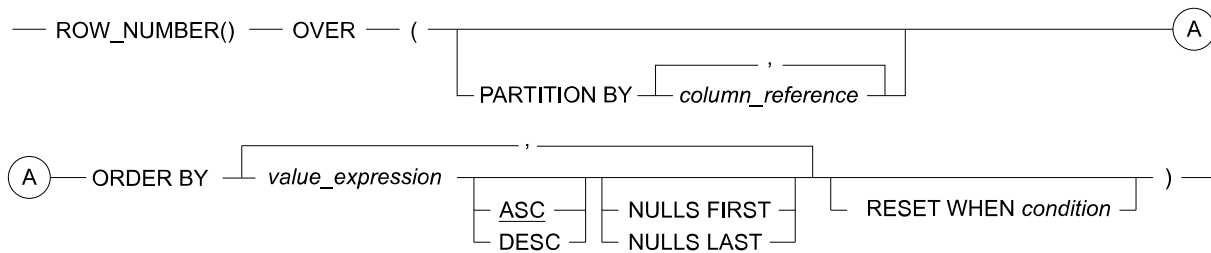
### Purpose

Returns the sequential row number, where the first row is number one, of the row within its window partition according to the window ordering of the window.

### Type

ANSI SQL:2011 window function.

## Syntax



## Syntax Elements

### OVER

Specifies how values are grouped, ordered, and considered when computing the cumulative, group, or moving function.

Values are grouped according to the `PARTITION BY` `BEGIN` and `RESET WHEN` clauses `END`, sorted according to the `ORDER BY` clause, and considered according to the aggregation group within the partition.

### PARTITION BY

The group or groups over which the function operates.

### ORDER BY

The order in which the values in a group or partition are sorted.

### ASC

Ascending sort order.

### DESC

Descending sort order.

## NULLS FIRST

NULL results are to be listed first.

## NULLS LAST

NULL results are to be listed last.

## RESET WHEN

The group, or groups, over which the function operates, depending on the evaluation of the specified condition. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there is no PARTITION BY or RESET WHEN clauses, then the entire result set, delivered by the FROM clause, constitutes a partition.

### *condition*

A conditional expression used to determine conditional partitioning. The condition in the RESET WHEN clause is equivalent in scope to the condition in a QUALIFY clause with the additional constraint that nested ordered analytical functions cannot specify a RESET WHEN clause. In addition, you cannot specify SELECT as a nested subquery within the condition.

The condition is applied to the rows in all designated window partitions to create sub-partitions within the particular window partitions.

For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

## ANSI Compliance

This statement is ANSI SQL:2011 compliant, but includes non-ANSI Teradata Database extensions.

## Window Aggregate Equivalent

```
ROW_NUMBER() OVER (PARTITION BY column
ORDER BY value
)
```

is equivalent to

```
COUNT(*) OVER (PARTITION BY column
ORDER BY value

ROWS UNBOUNDED PRECEDING).
```

## Example

To order salespersons based on sales within a sales region, the following SQL query might yield the following results.

```
SELECT ROW_NUMBER() OVER (PARTITION BY sales_region
                           ORDER BY sales_amount DESC),
sales_person, sales_region, sales_amount
FROM sales_table;
```

Row_Number()	sales_person	sales_region	sales_amount
-----	-----	-----	-----
1	Baker	East	100
2	Edwards	East	99
3	Davis	East	89
4	Adams	East	75
1	Garabaldi	West	100
2	Connors	West	99
3	Fine	West	99

## Related Topics

For more information, see:

- For more information, see “RESET WHEN Condition Rules” and the “QUALIFY Clause” in *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.
- For more information on COUNT, see [Window Aggregate Functions](#).



teradata.

**CFilter**

## **Association Analysis**

Teradata Vantage Analytics Workshop  
ADVANCED ILT

©2019 Teradata

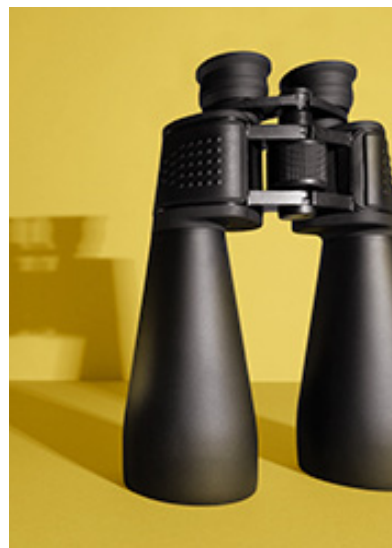
## Objectives

After completing this module, you will be able to:

- Describe what the **CFilter** functions does
- Describe typical use cases for **CFilter**
- Write **CFilter** queries
- Interpret the output of **CFilter** queries

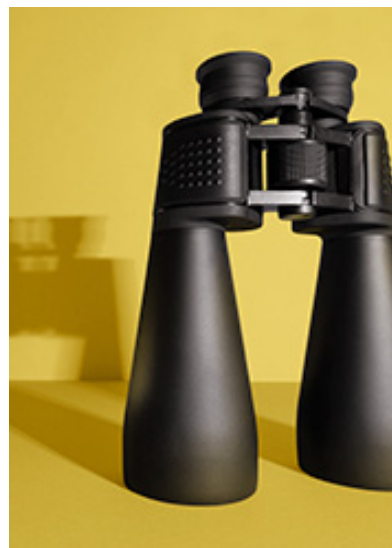
## Topics

- CFilter
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - Review



## Current Topic – CFilter Background Information

- **CFilter**
  - **Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)**
  - Labs
  - Review



## CFilter Description

- The **CFilter** function creates baskets (sets) of two-way interactions. The input is typically a set of purchase transaction records (e.g., what was bought) or Web page-view logs (e.g., what Web pages were visited)
- Each basket is a unique permutation of two-way interactions; i.e., *butter:eggs* would be displayed, as well as *eggs:butter*. Unlike **BasketGenerator**, though, with **CFilter** we are limited to building two-way interactions
- **CFilter** writes data to a user-specified output table. Unlike **BasketGenerator**, **CFilter** output includes a host of columns in an attempt to put context to the two-way interactions that it finds. These output columns can guide the end-user in understanding the relationships between the items in a pair and the pair in relation to all other pairs

The **CFilter** function creates baskets (sets) of two-way interactions. The input is typically a set of purchase transaction records (e.g., what was bought) or Web page-view logs (e.g., what Web pages were visited).

Each basket is a unique permutation of two-way interactions; i.e., *butter:eggs* would be displayed, as well as *eggs:butter*. Unlike **BasketGenerator**, though, with **CFilter** we are limited to building two-way interactions.

CFilter writes data to a user-specified output table. Unlike **BasketGenerator**, **CFilter** output includes a host of columns in an attempt to put context to the two-way interactions that it finds. These output columns can guide the end-user in understanding the relationships between the items in a pair and the pair in relation to all other pairs.

## CFilter Use Cases

The **CFilter** function deals with finding *two-way combinations* of entities (e.g., product domains purchased, Web pages visited, etc.) that co-occur at the defined level of analysis; e.g., household, unique\_trans\_id, etc. For example, it could be used to identify products that are bought together (*association analysis*, *basket analysis*, *affinity analysis*), such as peanut butter and jelly.

Many companies across a wide array of industries use collaborative filtering to 1) increase sales, 2) increase units-moved, 3) increase profit, 4) and increase relevancy to end consumers.

Before embarking on things such as offering free jelly if peanut butter is bought, as is always the case, it is recommended that you test your theories out in practice with strict adherence to treatment:control experimental-design principles so as to be able to quantify the effects of your efforts. For example, certain organizations may be reluctant to co-promote peanut and jelly, assuming that only one of the two needs to be promoted, as the other one will be bought anyway (at full-price, moreover); i.e., there will be an "erosion of margin" or a "loss of sales and profit" by co-promoting the affinity products. **Experiment to see what works and what doesn't work!**

**CFilter** provides numerous metrics to broaden your understanding of two-way interactions.

Such "Association Analysis" techniques have been used by numerous companies with great success to drive top- and bottom-line sales.

Take care to experiment find out what works and what doesn't.

## CFilter Use Cases

Following are some examples of how **CFilter** could be used:

- A **retailer** wishes to redesign the layout of its stores, so it runs **CFilter** to discover what product domains co-occur together; e.g., peanut butter and jelly. These products will be placed near one another on the shelf
- A **retailer** manufactures and carries a private-label peanut butter, but no private-label jelly; i.e., customers have no choice but to purchase the national-brand jelly—which is not as profitable as a private-label jelly would be. The retailer decides to start offering a private-label jelly for sale
- A **telecommunications** company wishes to offer particular bundlings of products and services to its clients, so it uses **CFilter** to discover which products/services have natural attractions to one another
- A **healthcare** company wishes to discover which ailments co-occur together on a patient-by-patient basis so as to help in future diagnoses of patients
- A **retailer** notices that there is a universe of customers who only purchase one-half of a significant affinity pairing, so they target such customers with the appropriate affinity-void offer

**CFilter** can be used by any business that wishes to understand the interplay of how products/services are purchased together.

## CFilter Workflow



- **Input Tables:** Data is read from a specified input table, views, or query.
- **CFilter:** The following arguments, at a minimum, are specified when the function is invoked.
  - **OutputTable**
  - **TargetColumns**
  - **JoinColumns**
- **Output table:** Data is written to the output table specified.

The **CFilter** function will read from a defined table, view, or query, and output the results to a table per its defined arguments.

## CFilter Syntax

```
SELECT * FROM CFilter (  
  ON { table | view | (query) } AS InputTable  
  OUT TABLE OutputTable (output_table)  
  USING  
    TargetColumns ({ 'target_column' | target_column_range }[,...])  
    JoinColumns ({ 'join_column' | join_column_range }[,...])  
    [ PartitionColumns ({ 'partition_column' | partition_column_range  
      }[,...]) ]  
    [ PartitionKey ('partition_key_column') ]  
    [ MaxDistinctItems (max_distinct_items) ]  
  ) AS alias;
```

Here, we show the base syntax for **CFilter**. Note that there are three required arguments:

- OutputTable
- TargetColumns
- JoinColumns

## CFilter Required Arguments (1 of 2)

The required arguments for the **CFilter** function are as follows:

**OutputTable**: Specify the name of the output table that the function creates. The table must not exist

**TargetColumns**: Specify the names of the input table columns that contain the data to filter

The required arguments for the CFilter function are as follows:

- **OutputTable**: Specify the name of the output table that the function creates. The table must not exist.
- **TargetColumns**: Specify the names of the input table columns that contain the data to filter.
- **JoinColumns**: Specify the names of join columns, which the function uses as follows:
  - The function uses the items in each join column to define groups of items listed in the input columns.
  - The function tries to identify items in each input column that often appear in the same group.

## CFilter Required Arguments (2 of 2)

**JoinColumns:** Specify the names of join columns, which the function uses as follows:

1. The function uses the items in each join column to define groups of items listed in the input columns
2. The function tries to identify items in each input column that often appear in the same group

For example, a join column might contain a list of sales transactions from a store, and the input column might contain each individual item purchased at the store. A sales transaction can include multiple items. For each sales transaction, the function tries to identify items that often appear in the same sales transaction (that is, items that are often purchased together).

The required arguments for the CFilter function are as follows:

- **OutputTable:** Specify the name of the output table that the function creates. The table must not exist.
- **TargetColumns:** Specify the names of the input table columns that contain the data to filter.
- **JoinColumns:** Specify the names of join columns, which the function uses as follows:
  - The function uses the items in each join column to define groups of items listed in the input columns.
  - The function tries to identify items in each input column that often appear in the same group.

## CFilter Optional Arguments (1 of 2)

The following **CFilter** arguments are optional.

**PartitionColumns**: [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns. **Default behavior**: The function treats the input data as belonging to one partition

**Note**: Specifying a column as both a `partition_column` and a `join_column` causes incorrect counts in partitions. This argument makes the function output nondeterministic unless each `partition_column` is unique in the group defined by `JoinColumns`

The following **CFilter** arguments are optional.

- **PartitionColumns**: [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns. Default behavior: The function treats the input data as belonging to one partition.
  - Note: Specifying a column as both a `partition_column` and a `join_column` causes incorrect counts in partitions. This argument makes the function output nondeterministic unless each `partition_column` is unique in the group defined by `JoinColumns`.
- **PartitionKey**: [Optional] Specify the name of the output column to use as the partition key. Default: 'col1\_item1'
- **MaxDistinctItems** [Optional] Specify the maximum size of the item set. Default: 100
  - Note: The function uses `max_item_set` to determine the size of the data structures it uses to accumulate intermediate results. If the number of distinct items in an `target_column` is greater than `max_item_set`, the function might report incorrect results without an error message.

## CFilter Optional Arguments (2 of 2)

**PartitionKey:** [Optional] Specify the name of the output column to use as the partition key. Default: 'col1\_item1'

**MaxDistinctItems** [Optional] Specify the maximum size of the item set. Default: 100

- **Note:** The function uses *max\_item\_set* to determine the size of the data structures it uses to accumulate intermediate results. If the number of distinct items in an *target\_column* is greater than *max\_item\_set*, the function might report incorrect results without an error message

The following **CFilter** arguments are optional.

- **PartitionColumns:** [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns. Default behavior: The function treats the input data as belonging to one partition.
  - Note: Specifying a column as both a *partition\_column* and a *join\_column* causes incorrect counts in partitions. This argument makes the function output nondeterministic unless each *partition\_column* is unique in the group defined by *JoinColumns*.
- **PartitionKey:** [Optional] Specify the name of the output column to use as the partition key. Default: 'col1\_item1'
- **MaxDistinctItems** [Optional] Specify the maximum size of the item set. Default: 100
  - Note: The function uses *max\_item\_set* to determine the size of the data structures it uses to accumulate intermediate results. If the number of distinct items in an *target\_column* is greater than *max\_item\_set*, the function might report incorrect results without an error message.

## CFilter Input Table Schema

Column	Data Type	Description
<i>target_column</i>	VARCHAR	Data to filter
<i>join_column</i>	Any	Column to join
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column to copy to output table. Used to partition input data and output table. Must not be a <i>join_column</i> . Must be unique in the group defined by <i>JoinColumns</i> , or function output is nondeterministic

**Note:** Input-table column names that you reference in your [CFilter](#) queries must be named using all lower-case letters.

This page displays the input table schema.

## CFilter Output Table Schema (1 of 5)

Column	Data Type	Description
<i>col1_item1</i>	VARCHAR	Name of item1
<i>col1_item2</i>	VARCHAR	Name of item2
<i>cntb</i>	INTEGER	Count of co-occurrence of both items in partition
<i>cnt1</i>	INTEGER	Count of occurrence of item1 in partition
<i>cnt2</i>	INTEGER	Count of occurrence of item2 in partition
<i>score</i>	DOUBLE PRECISION	Product of two conditional probabilities: $P(\{ \text{item2} \mid \text{item1} \}) * P(\{ \text{item1} \mid \text{item2} \})$  Preceding product equals following quotient:  $(cntb * cntb)/(cnt1 * cnt2)$

This page displays the output table schema.

## CFilter Output Table Schema (2 of 5)

Column	Data Type	Description
<i>support</i>	DOUBLE PRECISION	<p>Percentage of transactions in partition in which the two items co-occur, calculated with this formula:</p> $\text{cntb}/\text{tran\_cnt}$ <p>where <i>tran_cnt</i> is the number of transactions in the partition</p> <p>For example, if eggs and milk were purchased together 3 times in 5 transactions in the same store, and the data is partitioned by store, then the support value in the partition is <math>3/5 = 0.6 = 60\%</math></p>

This page displays the output table schema.

## CFilter Output Table Schema (3 of 5)

Column	Data Type	Description
<i>confidence</i>	DOUBLE PRECISION	<p>Percentage of transactions in partition in which <i>item1</i> occurs, in which <i>item2</i> also occurs, calculated with this formula:</p> <p><b>cntb/cnt1</b></p> <p>For example, if, in the same store, the number of times that a customer buys both milk (<i>item1</i>) and butter (<i>item2</i>) is 3 (<i>cntb</i>) and the number of times that a customer buys milk is 4 (<i>cnt1</i>), then the confidence that a person who buys milk will also buy butter is <math>3/4 = 0.75 = 75\%</math></p>

This page displays the output table schema.

## CFilter Output Table Schema (4 of 5)

Column	Data Type	Description
<i>lift</i>	DOUBLE PRECISION	<p>Ratio of observed support value to expected support value if <i>item1</i> and <i>item2</i> were independent; that is: <math>\text{support}(\text{item1 and item2}) / [\text{support}(\text{item1}) * \text{support}(\text{item2})]</math></p> <p>Value is calculated with this formula:</p> $(\text{cntb}/\text{tran\_cnt}) / [(\text{cnt1}/\text{tran\_cnt}) * (\text{cnt2}/\text{tran\_cnt})]$ <ul style="list-style-type: none"><li>• If Lift &gt; 1, the occurrence of <i>item1</i> or <i>item2</i> has a positive effect on the occurrence of the other items</li><li>• If Lift = 1, the occurrence of <i>item1</i> or <i>item2</i> has a no effect on the occurrence of the other items</li><li>• If Lift &lt; 1, the occurrence of <i>item1</i> or <i>item2</i> has a negative effect on the occurrence of the other items</li></ul>

This page displays the output table schema.

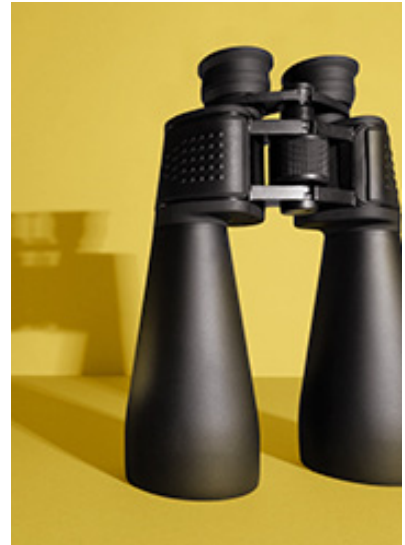
## CFilter Output Table Schema (5 of 5)

Column	Data Type	Description
<i>z_score</i>	DOUBLE PRECISION	Significance of co-occurrence, assuming that <i>cntb</i> follows a normal distribution, calculated with this formula:  $(\text{cntb} - \text{mean}(\text{cntb})) / \text{sd}(\text{cntb})$  If all <i>cntb</i> values are equal, then <i>sd(cntb)</i> is 0, and function does not calculate <i>zscore</i>

This page displays the output table schema.

## Current Topic – CFilter Labs

- **CFilter**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
- **Labs**
- Review





## Lab 01a – Understanding CFilter Output: Reviewing Input Data

teradata.

Data Table

```
SELECT *  
FROM bb_cfilter_test  
ORDER BY tranid, item;
```

- For this initial lab, we will look at a small data-set of transactions and discuss each of the fields that **CFilter** outputs, using only *required arguments*.
- Our transaction data appears to the right

	tranid	dt	storeid	region	item	sku	category
1	1	20100715	1	west	butter	2	dairy
2	1	20100715	1	west	eggs	3	dairy
3	1	19990715	1	west	flour	4	baking
4	1	20100715	1	west	milk	1	dairy
5	2	20100715	1	west	butter	2	dairy
6	2	20100715	1	west	eggs	3	dairy
7	2	20100715	1	west	milk	1	dairy
8	3	20100715	1	west	eggs	3	dairy
9	3	19990715	1	west	flour	4	baking
10	3	20100715	1	west	milk	1	dairy
11	4	20100715	1	west	butter	2	dairy
12	4	20100715	1	west	milk	1	dairy
13	5	20100715	2	west	butter	2	dairy
14	5	20100715	2	west	eggs	3	dairy
15	5	19990715	2	west	flour	4	baking
16	6	20100715	2	west	eggs	3	dairy
17	6	20100715	2	west	milk	1	dairy
18	7	20100715	2	west	eggs	3	dairy
19	7	19990715	2	west	flour	4	baking
20	8	20100715	3	west	butter	2	dairy
21	8	20100715	3	west	eggs	3	dairy
22	8	19990715	3	west	flour	4	baking

Here, we are scrutinizing the contents of a simple input table. The next many pages will go through various examples of running the **CFilter** function against this table and reviewing the resulting output.



## Lab 01b – Understanding CFilter Output: Reviewing Input Data

teradata.

Here, we are segregating fields of interest and sorting the data:

- We have eight distinct transactions
- We have four distinct items

```
SELECT tranid, item
FROM bb_cfilter_test
ORDER BY tranid, item;
```

```
SELECT item, tranid
FROM bb_cfilter_test
ORDER BY item, tranid;
```

**Eight transactions**

	tranid	item
1	1	butter
2	1	eggs
3	1	flour
4	1	milk
5	2	butter
6	2	eggs
7	2	milk
8	3	eggs
9	3	flour
10	3	milk
11	4	butter
12	4	milk
13	5	butter
14	5	eggs
15	5	flour
16	6	eggs
17	6	milk
18	7	eggs
19	7	flour
20	8	butter
21	8	eggs
22	8	flour

**Four items**

	item	tranid
1	butter	1
2	butter	2
3	butter	4
4	butter	5
5	butter	8
6	eggs	1
7	eggs	2
8	eggs	3
9	eggs	5
10	eggs	6
11	eggs	7
12	eggs	8
13	flour	1
14	flour	3
15	flour	5
16	flour	7
17	flour	8
18	milk	1
19	milk	2
20	milk	3
21	milk	4
22	milk	6

Here, we are scrutinizing the contents of a simple input table.

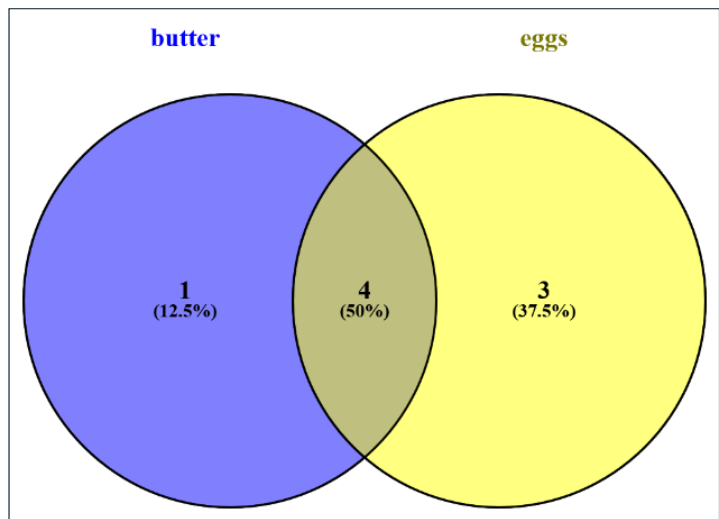


## Lab 01c – Understanding CFilter Output: Reviewing Input Data

teradata.

Here, we are viewing a Venn diagram that summarizes the transactions for *eggs* and/or *butter* from the previous page:

- *Butter* occurred in **5** transactions (1+4)
- *Eggs* occurred in **7** transactions (3+4)
- There were **8** transactions that contained *butter and/or eggs* (1+4+3)
- There were **4** transactions that contained both *eggs and butter*
- There was **1** transaction that contained *butter with no eggs*
- There were **3** transactions that contained *eggs with no butter*



Here, we are summarizing the domain of transactions that involved either butter or eggs or both.



## Lab 01d – Understanding CFilter Output: Running CFilter Query

teradata.

```
--must drop output table if exists
DROP TABLE bb_cfilter_test_output;

--run CFilter
SELECT *
FROM CFilter (
ON bb_cfilter_test AS InputTable
OUT TABLE OutputTable
(bb_cfilter_test_output)
USING
TargetColumns ('item')
JoinColumns ('tranid')
) AS my_alias;
```

Here, we are running **CFilter** against our input table. Note the required arguments:

- **OUT TABLE**: This is the table to which **CFilter** will write the results. We have defined **bb\_cfilter\_test\_output**
- **TargetColumns**: This is the column against which we would like to find co-occurrences. We have defined **item**; e.g., find co-occurrences of *butter* and *eggs*
- **JoinColumns**: This is the column that defines at what level we would like to find co-occurrences of items. We have defined **tranid**; e.g., find co-occurrences of *butter* and *eggs* if and only if they occurred *within the same transaction*

Here, we are running a simple **CFilter** query against our input data, utilizing only required arguments.

Note that the **OUT TABLE** must not exist or the query will fail; therefore, we first drop the defined **OUT TABLE**.



## Lab 01e – Understanding CFilter Output: Reviewing Output Table

teradata.

```
--interrogate output table
SELECT *
FROM bb_cfilter_test_output
ORDER BY col1_item1, col1_item2;
```

- Here, we are selecting all columns from our generated output table
- Unlike **BasketGenerator**, note that in **CFilter**, by default, *butter:eggs* is not equivalent to *eggs:butter*; i.e., all product-pairings appear in both directions

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...
2	butter	flour	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
3	butter	milk	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
4	eggs	butter	4	7	5	0.45714...	0.5	0.5714285...	0.9142...	0.522232...
5	eggs	flour	5	7	5	0.71428...	0.625	0.7142857...	1.1428...	1.566698...
6	eggs	milk	4	7	5	0.45714...	0.5	0.5714285...	0.9142...	0.522232...
7	flour	butter	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
8	flour	eggs	5	5	7	0.71428...	0.625	1.0	1.1428...	1.566698...
9	flour	milk	2	5	5	0.16	0.25	0.4	0.64	-1.56669...
10	milk	butter	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
11	milk	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...
12	milk	flour	2	5	5	0.16	0.25	0.4	0.64	-1.56669...

Here we are viewing the contents of the generated output table.

Note the following:

- **CFilter** generates all possible pairs. You will see an output row for **butter:eggs** and another output row **eggs:butter**.
- Out of the five scores to the right of the output, only **confidence** will differ between any two sister-rows.
- Unlike **BasketGenerator**, **CFilter** does not allow you to specify three-way, four-way, etc. interactions. **CFilter** calculates only two-way interactions.



## Lab 01f – Understanding CFilter Output: Reviewing Basic Output Columns

teradata.

Following is a discussion of the first five columns generated by **CFilter**. Also, note that **CFilter** only generates two-way interactions. It cannot generate multi-way interactions like **BasketGenerator** can.

- **col1\_item1**: The first item of the pair
- **col2\_item2**: The second item of the pair
- **cntb**: The number of transactions in which both items appeared
- **cnt1**: The number of transactions in which item1 appeared
- **cnt2**: The number of transactions in which item2 appeared

	col1_item1	col2_item2	cntb	cnt1	cnt2
1	butter	eggs	4	5	7
2	butter	flour	3	5	5
3	butter	milk	3	5	5
4	eggs	butter	4	7	5
5	eggs	flour	5	7	5
6	eggs	milk	4	7	5

Following are brief definitions of all output columns:

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01g – Understanding CFilter Output: Score

**Score** displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions). The formula for calculating score follows:

$$(\text{cntb} * \text{cntb}) / (\text{cnt1} * \text{cnt2})$$

which is equivalent to  $(\text{cntb} / \text{cnt1}) * (\text{cntb} / \text{cnt2})$ ; i.e.,

$$\frac{\text{trans with both}}{\text{trans with butter}} * \frac{\text{trans with both}}{\text{trans with eggs}} = \frac{4}{5} * \frac{4}{7} = 0.457142857$$

That is to say, there is roughly a 45% chance that if either *butter* or *eggs* is present in a transaction, then the other one will be present too. The highest possible score is **1**, implying that if one of them is present, there is a 100% probability that the other one will be as well.

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...

Following are brief definitions of all output columns:

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01h – Understanding CFilter Output: Support

teradata.

**Support** displays the percentage of all transactions that had both items present.

$$\text{cntb} / \text{total\_trans}$$

In the **CFilter** output, the value for **total\_trans** (total transaction count) is not displayed. Recall from an earlier slide, however, that our total count of distinct transactions was **8**. Therefore,

$$\frac{\text{trans with both}}{\text{total transaction count}} = \frac{4}{8} = 0.5$$

The larger **support** is, the more commonplace the co-occurrence was. A value of **1** would imply that every single transaction had the both items present. Here, we are seeing that 50% of all transactions had both *butter* and *eggs* present.

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	4	5	7	0.45714..	0.5	0.8	0.9142...	0.522232...

Following are brief definitions of all output columns:

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01i – Understanding CFilter Output: Confidence

teradata.

**Confidence** displays out of all transactions with **item1**, what percentage of them also had **item2**. You can think of this as the "crossover percent", with **item1** being the focal point.

$$\text{cntb} / \text{cnt1}$$

Therefore, we see that 80% of the transactions with *butter* also have *eggs* present.

$$\frac{\text{trans with both}}{\text{trans with butter}} = \frac{4}{5} = 0.8$$

If *butter* is present in a transaction, there is an 80% **confidence** that *eggs* will also be present in the same transaction. The larger that **confidence** is for *butter*, the more likely that *eggs* will also be present in the same transaction. A value of **1** would imply that every single transaction that had *butter* also had *eggs*.

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...

Following are brief definitions of all output columns:

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01j – Understanding CFilter Output: Lift

**Lift** displays how many times the actual **cntb** is from the expectation of what **cntb** should be. To calculate the expectation, we need to know how many total transactions there were. From an earlier slide, recall that the total transaction count was 8. The formula for **lift** is as follows:

$$(cntb / total\_trans) / ((cnt1 / total\_trans) * (cnt2 / total\_trans))$$

The formula for the expectation is  $(cnt1 * cnt2) / total\_trans$ . Therefore, the expectation is

$$\frac{trans\ with\ butter * trans\ with\ eggs}{total\ transactions} = \frac{5 * 7}{8} = 4.375$$

Our actual **cntb** was 4, and 4 is 0.9142 times the size 4.375—derived from  $\frac{4}{4.375} = 0.914285714$ . Using our original formula, we see that  $(4 / 8) / ((5 / 8) * (7 / 8)) = 0.914285714$ ; i.e., the actual co-occurrence of *butter* and *eggs* wasn't what mere chance would have suggested it should have been (it almost was, at roughly 91%).

- If **lift** > 1, then actual co-occurrence is greater than the expectation; i.e., *product attraction*
- If **lift** = 1, then actual co-occurrence is equal to the expectation; i.e., *product neutrality*
- If **lift** < 1, then actual co-occurrence is less than the expectation; i.e., *product repulsion*

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...

Following are brief definitions of all output columns:

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01k – Understanding CFilter Output: z\_score

teradata.

**Z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.

$$(\text{cntb} - \text{mean}(\text{cntb})) / \text{sd}(\text{cntb})$$

In our data-set, the average of **cntb** is **3.5** and the standard deviation of **cntb** is **0.957427108**. Therefore, for the co-occurrence of *butter* and *eggs*, we see the following:

$$\frac{\text{trans with both} - 3.5}{0.957427108} = \frac{4 - 3.5}{0.957427108} = 0.522232968$$

By its very nature, **z\_score** will be higher for common product-pairings and less for infrequent product-pairings; i.e., high-volume items will likely have a higher score. If co-promoting affinity pairings, you can think of **z\_score** as *volume potential*. The higher the score, the more overall volume potential. *Butter and eggs* is slightly more than half a standard deviation to the right of the mean relative to all product-pairings (it is slightly more commonplace).

- If **z\_score** > 0, then product-pairing occurs more frequently than typical product-pairing
- If **z\_score** < 0, then product-pairing occurs less frequently than typical product-pairing
- If **z\_score** = 0, then product-pairing occurs with the same frequency as the typical product-pairing

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...

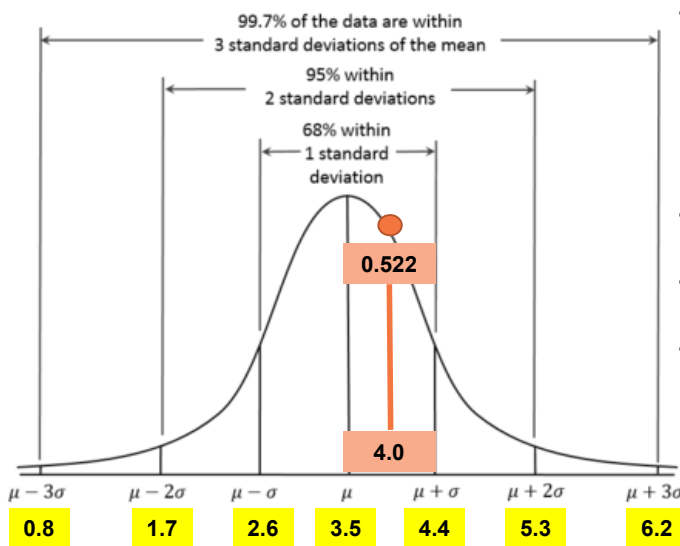
Following are brief definitions of all output columns:

For **z\_score**, note that if all **cntb** values are equal, then **sd(cntb)** is **0**, and the function does not calculate **z\_score** (returns **NULL**).

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01I – Understanding CFilter Output: z\_score



- In our data-set, the *average* of **cntb** is **3.5** and the *standard deviation* of **cntb** is roughly **0.9** (precision simplified here for illustrative purposes). *Butter and eggs* has a **cntb** value of **4**, or around half (**0.522**) a standard deviation to the right of the mean
- Each standard deviation to the left of the mean *decrements* by **0.9**; i.e., 2.6, 1.7, and 0.8
- Each standard deviation to the right of the mean *increments* by **0.9**; i.e., 4.4, 5.3, and 6.2
- Our **z\_score** for *butter and eggs* is roughly **+0.522** (shown in **orange** to the left), signifying that its **cntb** value of **4** is around half a standard deviation to the right of the **cntb** mean; i.e., its co-occurrence value is higher than the average, by around half a standard deviation

Following are brief definitions of all output columns:

For **z\_score**, note that if all **cntb** values are equal, then **sd(cntb)** is **0**, and the function does not calculate **z\_score** (returns **NULL**).

- **col1\_item1**: The first item of the pair.
- **col2\_item2**: The second item of the pair.
- **cntb**: The number of transactions in which both items appeared.
- **cnt1**: The number of transactions in which item1 appeared.
- **cnt2**: The number of transactions in which item2 appeared.
- **score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support**: Displays the percentage of all transactions that had both items present.
- **confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 01m – Understanding CFilter Output: Summary

teradata.

Following are some general guidelines on interpreting the output of **CFilter**.

- **No single output metric should necessarily be viewed in isolation:** View metrics as a collective
- **If volume is of most interest:** Focus on high **support** and/or high **z\_score** values. High values on these metrics put the emphasis on frequently-occurring product pairings. Products such as bananas, milk, eggs, bread, etc. by their sheer volume will have high **support** and high **z\_score** values when paired with other such similar products. Something like granola paired with organic yogurt will have lower **support** and **z\_score** values due to the relatively low volume of both of those products. If neither one is bought frequently, then, by extension, the raw co-occurrence, **support**, and **z\_score** values for the two will be relatively low as well
- **If product attraction is paramount:** Focus on high **score**, **confidence**, and **lift** values. High values on these metrics suggest that even though they may or may not be high-volume items, their attraction towards one another is strong; e.g., granola and organic yogurt
- Of course, you could focus on product pairings that have both *high volume and high product attraction*

Following are brief definitions of all output columns:

- **col1\_item1:** The first item of the pair.
- **col2\_item2:** The second item of the pair.
- **cntb:** The number of transactions in which both items appeared.
- **cnt1:** The number of transactions in which item1 appeared.
- **cnt2:** The number of transactions in which item2 appeared.
- **score:** Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support:** Displays the percentage of all transactions that had both items present.
- **confidence:** Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift:** Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 02a – Discovering Product Pairings

```
SELECT *  
FROM sales_transaction  
ORDER BY orderid, product;
```

Here, we are viewing a sub-set of rows from our input table. On the next page, we will run a **CFilter** query against this input table.

The columns of interest are as follows:

- **orderid**: The unique identifier of a transaction
- **product**: The product that was purchased

	orderid	orderdate	orderatv	region	customer se...	prd category	product
1	3	2010-10-13	6	nunavut	small business	office supplies	storage & or...
2	293	2012-10-01	49	nunavut	consumer	office supplies	appliances
3	293	2012-10-01	27	nunavut	consumer	office supplies	binders and ...
4	483	2011-07-10	30	nunavut	corporate	technology	telephones a...
5	515	2010-08-28	19	nunavut	consumer	office supplies	appliances
6	515	2010-08-28	21	nunavut	consumer	furniture	office furnish...
7	613	2011-06-17	12	nunavut	corporate	office supplies	binders and ...
8	613	2011-06-17	22	nunavut	corporate	office supplies	storage & or...
9	643	2011-03-24	21	nunavut	corporate	office supplies	storage & or...

Here, we are familiarizing ourselves with the input table.



## Lab 02b – Discovering Product Pairings

### DROP TABLE

```
bb_borre_cfilter_output;
```

```
SELECT * FROM CFilter (  
ON sales_transaction AS  
InputTable  
OUT TABLE OutputTable  
(bb_borre_cfilter_output)  
USING  
TargetColumns ('product')  
JoinColumns ('orderid')  
) AS my_alias;
```

Here, we are running our **CFilter** query. Note the following:

- We first drop our **OUT TABLE**
- We have defined the **product** column as our **TargetColumns** value. This is the product that was purchased
- We have defined **orderid** as our **JoinColumns** value. This is the unique identifier of a transaction

The end-result of these arguments is that we are seeking our *product-pairings* that occurred *within the same transaction*

Here, we are running our query. Note that we first drop our **OUT TABLE** (if it exists).



## Lab 02c – Discovering Product Pairings

- Below, we are selecting our **CFilter** results from our defined output table
- Data is displayed for product-pairings that occurred within the same transactions
- Recall that each distinct pairing will occupy its own row; *labels:envelopes* will be in one row, and *envelopes:labels* will be in another row
- Refer to the **Notes** below for a brief definition of each column in the output

```
SELECT *  
FROM bb_borre_cfilter_output;
```

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	labels	envelopes	1	23	11	0.003952569...	0.002932551...	0.043478260...	1.347826086...	-0.966010128...
2	storage & or...	tables	3	36	14	0.017857142...	0.008797653...	0.083333333...	2.029761904...	0.583852275...
3	tables	labels	1	14	23	0.003105590...	0.002932551...	0.071428571...	1.059006211...	-0.966010128...
4	paper	labels	3	56	23	0.006987577...	0.008797653...	0.053571428...	0.794254658...	0.583852275...
5	envelopes	labels	1	11	23	0.003952569...	0.002932551...	0.090909090...	1.347826086...	-0.966010128...
6	binders and ...	tables	2	57	14	0.005012531...	0.005865102...	0.035087719...	0.854636591...	-0.191078926...
7	labels	tables	1	23	14	0.003105590...	0.002932551...	0.043478260...	1.059006211...	-0.966010128...
8	rubber bands	labels	1	12	23	0.003623188...	0.002932551...	0.083333333...	1.235507246...	-0.966010128...
9	labels	telephones a...	3	23	47	0.008325624...	0.008797653...	0.130434782...	0.946345975...	0.583852275...

Following are brief definitions of all output columns:

- col1\_item1**: The first item of the pair.
- col2\_item2**: The second item of the pair.
- cntb**: The number of transactions in which both items appeared.
- cnt1**: The number of transactions in which item1 appeared.
- cnt2**: The number of transactions in which item2 appeared.
- score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- support**: Displays the percentage of all transactions that had both items present.
- confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 02d – Discovering Product Pairings

- Recall that **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value
- Here, we are sorting our data by **z\_score** DESC. What do you notice?
- The **support** value displays the percentage of all transactions that had both items present. What happens when we sort by **support** DESC?

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	computer pe...	paper	6	38	56	0.016917293...	0.017595307...	0.157894736...	0.961466165...	2.908645882...
2	paper	computer pe...	6	56	38	0.016917293...	0.017595307...	0.107142857...	0.961466165...	2.908645882...
3	office furnish...	binders and ...	6	48	57	0.013157894...	0.017595307...	0.125	0.747807017...	2.908645882...
4	binders and ...	office furnish...	6	57	48	0.013157894...	0.017595307...	0.105263157...	0.747807017...	2.908645882...
5	storage & or...	paper	5	36	56	0.012400793...	0.014662756...	0.138888888...	0.845734126...	2.133714680...
6	telephones a...	storage & or...	5	47	36	0.014775413...	0.014662756...	0.106382978...	1.007683215...	2.133714680...
7	storage & or...	telephones a...	5	36	47	0.014775413...	0.014662756...	0.138888888...	1.007683215...	2.133714680...
8	paper	storage & or...	5	56	36	0.012400793...	0.014662756...	0.089285714...	0.845734126...	2.133714680...
9	labels	pens & art su...	4	23	29	0.023988005...	0.011730205...	0.173913043...	2.044977511...	1.358783477...
10	pens & art su...	labels	4	29	23	0.023988005...	0.011730205...	0.137931034...	2.044977511...	1.358783477...

Following are brief definitions of all output columns:

- col1\_item1**: The first item of the pair.
- col2\_item2**: The second item of the pair.
- cntb**: The number of transactions in which both items appeared.
- cnt1**: The number of transactions in which item1 appeared.
- cnt2**: The number of transactions in which item2 appeared.
- score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- support**: Displays the percentage of all transactions that had both items present.
- confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 02e – Discovering Product Pairings

- Recall that **lift** displays how many times the actual **cntb** is from the expectation of what **cntb** should be
- Here, we are sorting our data by **lift** DESC. What do you notice? What does a high **lift** value, but low **z\_score** value represent?

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	copiers and f...	storage & or...	2	4	36	0.027777777...	0.005865102...	0.5	4.736111111...	-0.191078926...
2	storage & or...	copiers and f...	2	36	4	0.027777777...	0.005865102...	0.055555555...	4.736111111...	-0.191078926...
3	envelopes	office machi...	2	11	24	0.015151515...	0.005865102...	0.181818181...	2.583333333...	-0.191078926...
4	office machi...	envelopes	2	24	11	0.015151515...	0.005865102...	0.083333333...	2.583333333...	-0.191078926...
5	labels	pens & art su...	4	23	29	0.023988005...	0.011730205...	0.173913043...	2.044977511...	1.358783477...
6	pens & art su...	labels	4	29	23	0.023988005...	0.011730205...	0.137931034...	2.044977511...	1.358783477...
7	tables	storage & or...	3	14	36	0.017857142...	0.008797653...	0.214285714...	2.029761904...	0.583852275...
8	rubber bands	paper	4	12	56	0.023809523...	0.011730205...	0.333333333...	2.029761904...	1.358783477...
9	paper	rubber bands	4	56	12	0.023809523...	0.011730205...	0.071428571...	2.029761904...	1.358783477...
10	storage & or...	tables	3	36	14	0.017857142...	0.008797653...	0.083333333...	2.029761904...	0.583852275...

Following are brief definitions of all output columns:

- col1\_item1**: The first item of the pair.
- col2\_item2**: The second item of the pair.
- cntb**: The number of transactions in which both items appeared.
- cnt1**: The number of transactions in which item1 appeared.
- cnt2**: The number of transactions in which item2 appeared.
- score**: Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- support**: Displays the percentage of all transactions that had both items present.
- confidence**: Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- lift**: Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 03a – MaxDistinctItems

```
--must drop output table if exists
DROP TABLE bb_cfilter_test_output2;

--run CFilter
SELECT *
FROM CFilter (
ON bb_cfilter_test AS InputTable
OUT TABLE OutputTable
(bb_cfilter_test_output2)
USING
TargetColumns ('item')
JoinColumns ('tranid')
MaxDistinctItems (3)
) AS my_alias;
```

- Here, we are running another **CFilter** query against our input table used in Lab 01. Recall that this input table has four distinct items (*butter, eggs, flour, milk*)
- In our query here, we have specified a value of **MaxDistinctItems (3)**
- **MaxDistinctItems**: [Optional] Specify the maximum size of the item set. Default: **100**
  - **Note:** The function uses `max_item_set` to determine the size of the data structures it uses to accumulate intermediate results. If the number of distinct items in a `target_column` is greater than `max_item_set`, *the function might report incorrect results without an error message*

The default is **MaxDistinctItems (100)**. If your data contains more than **100** distinct entities for which you are trying to discover pairings, make sure to set **MaxDistinctItems** appropriately; i.e., greater than or equal to the number of distinct entities.



## Lab 03b – MaxDistinctItems

- Below are the results of our **CFilter** query from Lab 01 compared to our results from this lab, in which we specified a value of **MaxDistinctItems (3)**
- Note the highlighted differences
- The lesson here is **know your data**. The default **MaxDistinctItems** is **100**. If your data contains more than 100 distinct items, make sure to set the **MaxDistinctItems** argument appropriately

Output from Lab 01 (correct)

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...
2	butter	flour	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
3	butter	milk	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
4	eggs	butter	4	7	5	0.45714...	0.5	0.5714285...	0.9142...	0.522232...
5	eggs	flour	5	7	5	0.71428...	0.625	0.7142857...	1.1428...	1.566698...
6	eggs	milk	4	7	5	0.45714...	0.5	0.5714285...	0.9142...	0.522232...
7	flour	butter	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
8	flour	eggs	5	5	7	0.71428...	0.625	1.0	1.1428...	1.566698...
9	flour	milk	2	5	5	0.16	0.25	0.4	0.64	-1.56669...
10	milk	butter	3	5	5	0.36	0.375	0.6	0.96	-0.52223...
11	milk	eggs	4	5	7	0.45714...	0.5	0.8	0.9142...	0.522232...
12	milk	flour	2	5	5	0.16	0.25	0.4	0.64	-1.56669...

Output from this lab (incorrect)

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	butter	eggs	3	5	7	0.25714...	0.375	0.6	0.68571...	0.5222329...
2	butter	flour	2	5	5	0.16	0.25	0.4	0.64	-0.522232...
3	butter	milk	2	5	5	0.16	0.25	0.4	0.64	-0.522232...
4	eggs	butter	3	7	5	0.25714...	0.375	0.4285714...	0.68571...	0.5222329...
5	eggs	flour	4	7	5	0.45714...	0.5	0.5714285...	0.91428...	1.5666989...
6	eggs	milk	3	7	5	0.25714...	0.375	0.4285714...	0.68571...	0.5222329...
7	flour	butter	2	5	5	0.16	0.25	0.4	0.64	-0.522232...
8	flour	eggs	4	5	7	0.45714...	0.5	0.8	0.91428...	1.5666989...
9	flour	milk	1	5	5	0.04	0.125	0.2	0.32	-1.566698...
10	milk	butter	2	5	5	0.16	0.25	0.4	0.64	-0.522232...
11	milk	eggs	3	5	7	0.25714...	0.375	0.6	0.68571...	0.5222329...
12	milk	flour	1	5	5	0.04	0.125	0.2	0.32	-1.566698...

The default is **MaxDistinctItems (100)**. If your data contains more than **100** distinct entities for which you are trying to discover pairings, make sure to set **MaxDistinctItems** appropriately; i.e., greater than or equal to the number of distinct entities in your data.



## Lab 04a – JoinColumns

```
SELECT *  
FROM bb_basket01  
ORDER BY  
hh_id, trans_id, prod;
```

	hh_id	trans_id	prod
1	1	1	peanut butter
2	1	2	jelly
3	2	3	cheese
4	2	4	crackers
5	3	5	eggs
6	3	6	bacon
7	4	7	bagels
8	4	7	cream cheese

- Here, we view the contents of our input table
- Note that there are **4** distinct households and **7** distinct transactions
- Only **trans\_id 7** had more than one item present
- All households except **hh\_id 4** had only single-item transactions
- On the following pages, we will show examples of specifying different **JoinColumns** values when we run our **CFilter** function against our input data-set
- We will specify the following:
  - **JoinColumns ('hh\_id')**
  - **JoinColumns ('trans\_id')**

Here, we are familiarizing ourselves with the input table.

**JoinColumns:** Specify the names of join columns, which the function uses as follows:

1. The function uses the items in each join column to define groups of items listed in the input columns.
2. The function tries to identify items in each input column that often appear in the same group.

For example, a join column might contain a list of sales transactions from a store, and the input column might contain each individual item purchased at the store. A sales transaction can include multiple items. For each sales transaction, the function tries to identify items that often appear in the same sales transaction (that is, items that are often purchased together).



## Lab 04b – JoinColumns

```
--drop OUT TABLE
DROP TABLE bb_cfilter_trans;

SELECT *
FROM CFilter (
ON bb_basket01 AS InputTable
OUT TABLE OutputTable
(bb_cfilter_trans)
USING
TargetColumns ('prod')
JoinColumns ('trans_id')
) AS my_alias;

SELECT *
FROM bb_cfilter_trans
ORDER BY col1_item1,
col1_item2;
```

- Here, we are specifying **JoinColumns ('trans\_id')**
- This causes the function to seek out product-pairings that occurred within the same trans\_id (transaction)
- Since there was only one **trans\_id** in our source data that had more than one product present (**trans\_id 7**), only two rows are returned

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	bagels	cream cheese	1	1	1	1.0	0.142857...	1.0	7.0	null
2	cream cheese	bagels	1	1	1	1.0	0.142857...	1.0	7.0	null

Here, we are specifying **JoinColumns ('trans\_id')**. This causes the function to seek out product-pairings that occurred within the same **trans\_id** (transaction).

**JoinColumns:** Specify the names of join columns, which the function uses as follows:

1. The function uses the items in each join column to define groups of items listed in the input columns.
2. The function tries to identify items in each input column that often appear in the same group.

For example, a join column might contain a list of sales transactions from a store, and the input column might contain each individual item purchased at the store. A sales transaction can include multiple items. For each sales transaction, the function tries to identify items that often appear in the same sales transaction (that is, items that are often purchased together).



## Lab 04c – JoinColumns

```
DROP TABLE bb_cfilter_hh;

SELECT *
FROM CFilter (
ON bb_basket01 AS InputTable
OUT TABLE OutputTable
(bb_cfilter_hh)
USING
TargetColumns ('prod')
JoinColumns ('hh_id')
) AS my_alias;

SELECT *
FROM bb_cfilter_hh
ORDER BY col1_item1,
col1_item2;
```

- Here, we are specifying **JoinColumns ('hh\_id')**
- This causes the function to seek out product-pairings that were purchased by the same hh\_id (household)
- Since every household purchased two products in our source data, multiple rows are returned

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	bacon	eggs	1	1	1	1.0	0.25	1.0	4.0	null
2	bagels	cream cheese	1	1	1	1.0	0.25	1.0	4.0	null
3	cheese	crackers	1	1	1	1.0	0.25	1.0	4.0	null
4	crackers	cheese	1	1	1	1.0	0.25	1.0	4.0	null
5	cream cheese	bagels	1	1	1	1.0	0.25	1.0	4.0	null
6	eggs	bacon	1	1	1	1.0	0.25	1.0	4.0	null
7	jelly	peanut butter	1	1	1	1.0	0.25	1.0	4.0	null
8	peanut butter	jelly	1	1	1	1.0	0.25	1.0	4.0	null

Here, we are specifying **JoinColumns ('hh\_id')**. This causes the function to seek out product-pairings that were purchased by the same **hh\_id** (household).

**JoinColumns:** Specify the names of join columns, which the function uses as follows:

1. The function uses the items in each join column to define groups of items listed in the input columns.
2. The function tries to identify items in each input column that often appear in the same group.

For example, a join column might contain a list of sales transactions from a store, and the input column might contain each individual item purchased at the store. A sales transaction can include multiple items. For each sales transaction, the function tries to identify items that often appear in the same sales transaction (that is, items that are often purchased together).



## Lab 05a – PartitionColumns

```
SELECT *  
FROM bb_trans11  
ORDER BY hh_id, trans_id;
```

	hh_id	region	trans_id	prod
1	a	miami	1	black beans
2	a	miami	1	rice
3	a	miami	3	yuca
4	a	miami	3	platanos
5	b	new york	2	bagels
6	b	new york	2	cream cheese
7	b	new york	4	hot dogs
8	b	new york	4	mustard

- Here, we view the contents of our input table
- Note that transactions occurred in two different regions: Miami and New York
- On the following pages, we will show an example of how the **PartitionColumns** argument can be used
- **PartitionColumns**: [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns
- **PartitionColumns** is especially useful if you wish to have **CFilter** output calculated independently for different *timeframes*, *geographies*, *customer segments*, etc.

Here, we are familiarizing ourselves with the input table.

**PartitionColumns**: [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns.

**Note:**

- Specifying a column as both an **partition\_column** and a **join\_column** causes incorrect counts in partitions.
- This argument makes the function output nondeterministic unless each **partition\_column** is unique in the group defined by **JoinColumns**.



## Lab 05b – PartitionColumns

- Here, we are not using the optional **PartitionColumns** argument
- All product-pairings and their corresponding metrics are *calculated globally*

```

DROP TABLE bb_cfilter_wop;

SELECT *
FROM CFilter (ON bb_trans11 AS InputTable
OUT TABLE OutputTable (bb_cfilter_wop)
USING
TargetColumns ('prod')
JoinColumns ('trans_id')
) AS my_alias;

SELECT *
FROM bb_cfilter_wop
ORDER BY col1_item1, col1_item2;

```

Output

	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	bagels	cream cheese	1	1	1	1.0	0.25	1.0	4.0	null
2	black beans	rice	1	1	1	1.0	0.25	1.0	4.0	null
3	cream cheese	bagels	1	1	1	1.0	0.25	1.0	4.0	null
4	hot dogs	mustard	1	1	1	1.0	0.25	1.0	4.0	null
5	mustard	hot dogs	1	1	1	1.0	0.25	1.0	4.0	null
6	platanos	yuca	1	1	1	1.0	0.25	1.0	4.0	null
7	rice	black beans	1	1	1	1.0	0.25	1.0	4.0	null
8	yuca	platanos	1	1	1	1.0	0.25	1.0	4.0	null

Here, we are running our query without **PartitionColumns**.

**PartitionColumns:** [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns.

**Note:**

- Specifying a column as both an **partition\_column** and a **join\_column** causes incorrect counts in partitions.
- This argument makes the function output nondeterministic unless each **partition\_column** is unique in the group defined by **JoinColumns**.

Following are brief definitions of all output columns:

- **col1\_item1:** The first item of the pair.
- **col2\_item2:** The second item of the pair.
- **cntb:** The number of transactions in which both items appeared.
- **cnt1:** The number of transactions in which item1 appeared.
- **cnt2:** The number of transactions in which item2 appeared.
- **score:** Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support:** Displays the percentage of all transactions that had both items present.
- **confidence:** Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift:** Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.



## Lab 05c – PartitionColumns

- Here, we are using the optional **PartitionColumns** argument
- All product-pairings and their corresponding metrics are *calculated by region*

```
DROP TABLE bb_cfilter_wp;
```

```
SELECT *  
FROM CFilter (  
ON bb_trans11 AS InputTable  
OUT TABLE OutputTable (bb_cfilter_wp)  
USING
```

```
TargetColumns ('prod')  
JoinColumns ('trans_id')  
PartitionColumns ('region')  
) AS my_alias;
```

```
SELECT *  
FROM bb_cfilter_wp  
ORDER BY region, col1_item1, col1_item2;
```

Output

	region	col1_item1	col1_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
1	miami	black beans	rice	1	1	1	1.0	0.5	1.0	2.0	null
2	miami	platanos	yuca	1	1	1	1.0	0.5	1.0	2.0	null
3	miami	rice	black beans	1	1	1	1.0	0.5	1.0	2.0	null
4	miami	yuca	platanos	1	1	1	1.0	0.5	1.0	2.0	null
5	new york	bagels	cream cheese	1	1	1	1.0	0.5	1.0	2.0	null
6	new york	cream cheese	bagels	1	1	1	1.0	0.5	1.0	2.0	null
7	new york	hot dogs	mustard	1	1	1	1.0	0.5	1.0	2.0	null
8	new york	mustard	hot dogs	1	1	1	1.0	0.5	1.0	2.0	null

Here, we are running our query with the optional **PartitionColumns** argument.

**PartitionColumns:** [Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns.

**Note:**

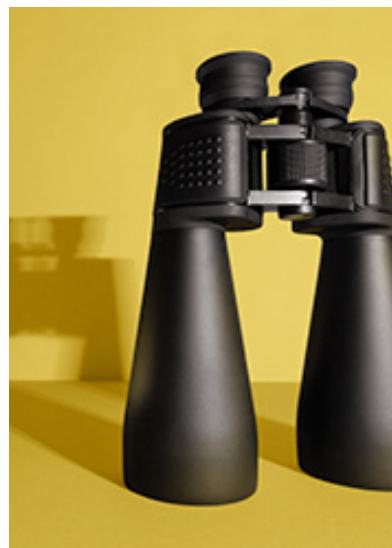
- Specifying a column as both an **partition\_column** and a **join\_column** causes incorrect counts in partitions.
- This argument makes the function output nondeterministic unless each **partition\_column** is unique in the group defined by **JoinColumns**.

Following are brief definitions of all output columns:

- **col1\_item1:** The first item of the pair.
- **col2\_item2:** The second item of the pair.
- **cntb:** The number of transactions in which both items appeared.
- **cnt1:** The number of transactions in which item1 appeared.
- **cnt2:** The number of transactions in which item2 appeared.
- **score:** Displays the chance of both products co-occurring together in the same transaction--ONLY considering the totality of transactions that had both (as opposed to the totality of all transactions).
- **support:** Displays the percentage of all transactions that had both items present.
- **confidence:** Displays out of all transactions with item1, what percentage of them also had item2. You can think of this as the "crossover percent", with item1 being the focal point.
- **lift:** Displays how many times the actual **cntb** is from the expectation of what **cntb** should be.
- **z\_score** displays how many standard deviations the **cntb** value lies from the average **cntb** value.

## Current Topic – CFilter Review

- **CFilter**
  - Background Information (Description, Use Cases, Workflow, Syntax, Required Arguments, Optional Arguments, Input Table Schema, Output Table Schema)
  - Labs
  - **Review**





## Hackathon: Product Combination Metrics

The following exercise is intended to provide you with further practice on using the **CFilter** function. There is no single “right” or “wrong” approach. The intent is for you to become comfortable writing queries that use **CFilter**

1. Run a **CFilter** query on the **sales\_transaction** table. Make sure the **CFilter** runs its logic against each **region:customer\_segment** combination. What rows have the highest **lift**? How often did the highest co-occurrence occur? Etc
2. Run a **CFilter** query on the **bb\_sales\_fact** table to find product-pairings that customers buy (i.e., don't need to be in the same transaction). What rows have the highest **lift**? What about rows with the most co-occurrences? Etc. What would you need to do to find product-pairings that co-occurred *within the same transaction*?



sales_transaction	
Columns	
orderid	[INTEGER Nullable]
orderdate	[VARCHAR(1024) Nullable]
orderqty	[INTEGER Nullable]
region	[VARCHAR(1024) Nullable]
customer_segment	[VARCHAR(1024) Nullable]
prd_category	[VARCHAR(1024) Nullable]
product	[VARCHAR(1024) Nullable]

bb_sales_fact	
Columns	
sales_date	[DATE Nullable]
customer_id	[INTEGER Nullable]
store_id	[INTEGER Nullable]
basket_id	[BIGINT Nullable]
product_id	[INTEGER Nullable]
sales_quantity	[INTEGER Nullable]
discount_amount	[FLOAT Nullable]
product_name	[VARCHAR(255) Nullable]
product_category_name	[VARCHAR(255) Nullable]
retail_price	[DECIMAL(9, 2) Nullable]
unit_cost	[DECIMAL(9, 2) Nullable]

In this “free-form” exercise, the intent is to get you to write your own **CFilter** query(ies) so as to become more comfortable with the syntax.



# Hackathon: Product Combination Metrics (Answers)

teradata.

region	customer se...	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
nunavut	home office	bookcases	tables	1	1	1	1.0	0.071428...	1.0	14.0	null
nunavut	home office	tables	bookcases	1	1	1	1.0	0.071428...	1.0	14.0	null
nunavut	corporate	office furnish...	bookcases	1	2	1	0.5	0.04	0.5	12.5	null

	region	customer se...	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	northwest ter...	corporate	telephones a...	storage & or...	3	22	12	0.034090909...	0.023622047...	0.1363636...	1.44318...	2.1896393...
2	northwest ter...	corporate	binders and ...	telephones a...	3	23	22	0.017786561...	0.023622047...	0.1304347...	0.75296...	2.1896393...
3	northwest ter...	corporate	binders and ...	rubber bands	3	23	5	0.078260869...	0.023622047...	0.1304347...	3.31304...	2.1896393...
4	northwest ter...	corporate	telephones a...	binders and ...	3	22	23	0.017786561...	0.023622047...	0.1363636...	0.75296...	2.1896393...
5	northwest ter...	corporate	binders and ...	office furnish...	3	23	17	0.023017902...	0.023622047...	0.1304347...	0.97442...	2.1896393...

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confidence	lift	z score
1	Milkshake	Chocolate Tr...	2666	3386	3349	0.626784414...	0.510434616...	0.787359716...	1.22794260...	0.024203743...
2	Chocolate Tr...	Milkshake	2666	3349	3386	0.626784414...	0.510434616...	0.796058524...	1.22794260...	0.024203743...
3	Cokodok	Geplak	2651	3375	3349	0.621771098...	0.507562703...	0.785481481...	1.22501337...	-0.607920061...
4	Geplak	Cokodok	2651	3349	3375	0.621771098...	0.507562703...	0.791579575...	1.22501337...	-0.607920061...
5	Deep-fried T...	Bugles	2703	3397	3394	0.633701458...	0.517518667...	0.795702090...	1.22449971...	1.583442463...

	col1 item1	col1 item2	cntb	cnt1	cnt2	score	support	confide...	lift	z score
1	Tuna Snacks	Peaches	2750	3464	3450	0.6328028...	0.526517...	0.793879...	1.2018651...	3.5640970...
2	Peaches	Tuna Snacks	2750	3450	3464	0.6328028...	0.526517...	0.797101...	1.2018651...	3.5640970...
3	Almonds	Tuna Snacks	2747	3438	3464	0.6336268...	0.525942...	0.799011...	1.2047444...	3.4376722...
4	Tuna Snacks	Almonds	2747	3464	3438	0.6336268...	0.525942...	0.793013...	1.2047444...	3.4376722...
5	Tuna Snacks	Blueberries	2745	3464	3449	0.6306866...	0.525560...	0.792436...	1.2000277...	3.3533891...

In this “free-form” exercise,, the intent is to get you to write your own **CFilter** query(ies) so as to become more comfortable with the syntax.

## Game Time! CFilter Hoops!

[Click here to start!](#)



This game, containing review questions, reinforces the module objectives.

## Summary

In this module, you learned how to:

- Describe what the **CFilter** function does
- Describe typical use cases for **CFilter**
- Write **CFilter** queries
- Interpret the output of **CFilter** queries

# CFilter (ML Engine)

A typical input table for the CFilter function is a set of sales transactions, with a column of purchased items and a column of something by which to group the purchased items; for example, a transaction id or time stamp.

The CFilter function calculates several statistical measures of how likely each pair of items is to be purchased together. You can use CFilter output as input to the WSRecommender function, which performs item-based, collaborative filtering, using a weighted-sum algorithm, to make recommendations.

## CFilter Syntax

### Version 1.13

```
SELECT * FROM CFilter (  
  ON { table | view | (query) } AS InputTable  
  OUT TABLE OutputTable (output_table)  
  USING  
  TargetColumns ({ 'target_column' | target_column_range }[,...])  
  JoinColumns ({ 'join_column' | join_column_range }[,...])  
  [ PartitionColumns ({ 'partition_column' | partition_column_range }[,...]) ]  
  [ PartitionKey ('partition_key_column') ]  
  [ MaxDistinctItems (max_distinct_items) ]  
) AS alias;
```

### Related Information:

[Column Specification Syntax Elements](#)

## CFilter Syntax Elements

### OutputTable

Specify the name of the output table that the function creates. The table must not exist.

### TargetColumns

Specify the names of the InputTable columns that contain the data to filter.

### JoinColumns

Specify the names of join columns, which the function uses as follows:

1. The function uses the items in each join column to define groups of items listed in the input columns.
2. The function tries to identify items in each input column that often appear in the same group.

For example, a join column might contain a list of sales transactions from a store, and the input column might contain each individual item purchased at the store. A sales transaction can include multiple items. For each sales transaction, the function tries to identify items that often appear in the same sales transaction (that is, items that are often purchased together).

### PartitionColumns

[Optional] Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns.

Specifying a column as both an *partition\_column* and a *join\_column* causes incorrect counts in partitions.

This syntax element makes the function output nondeterministic unless each *partition\_column* is unique in the group defined by JoinColumns (for more information, see [Nondeterministic Results and UniqueID Syntax Element](#)).

Default behavior: The function treats the input data as belonging to one partition.

### PartitionKey

[Optional] Specify the name of the output column to use as the partition key.

Default: 'col1\_item1'

### MaxDistinctItems

[Optional] Specify the maximum size of the item set.

The function uses *max\_item\_set* to determine the size of the data structures it uses to accumulate intermediate results. If the number of distinct items in an *target\_column* is greater than *max\_item\_set*, the function might report incorrect results without an error message.

Default: 100

## CFilter Input

### InputTable Schema

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>target_column</i>	VARCHAR	Data to filter.
<i>join_column</i>	Any	Column to join.
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column to copy to output table. Used to partition input data and output table. Must not be a <i>join_column</i> . Must be unique in the group defined by JoinColumns, or function output is nondeterministic (for more information, see <a href="#">Nondeterministic Results and UniqueID Syntax Element</a> ).

## CFilter Output

### Output Message Schema

Column	Data Type	Description
message	VARCHAR	Reports that output table was created successfully.

### OutputTable Schema

Output is nondeterministic unless each *add\_column* is unique in the group defined by JoinColumns (for more information, see [Nondeterministic Results and UniqueID Syntax Element](#)).

Column	Data Type	Description
col1_item1	VARCHAR	Name of item1.
col1_item2	VARCHAR	Name of item2.
cntb	INTEGER	Count of co-occurrence of both items in partition.
cnt1	INTEGER	Count of occurrence of item1 in partition.
cnt2	INTEGER	Count of occurrence of item2 in partition.
score	DOUBLE PRECISION	Product of two conditional probabilities: $P(\{ \text{item2} \mid \text{item1} \}) * P(\{ \text{item1} \mid \text{item2} \})$ Preceding product equals following quotient: $(cntb * cntb) / (cnt1 * cnt2)$
support	DOUBLE PRECISION	Percentage of transactions in partition in which the two items co-occur, calculated with this formula: $cntb / tran\_cnt$ where <i>tran_cnt</i> is the number of transactions in the partition. For example, if eggs and milk were purchased together 3 times in 5 transactions in the same store, and the data is partitioned by store, then the support value in the partition is $3/5 = 0.6 = 60\%$ .
confidence	DOUBLE PRECISION	Percentage of transactions in partition in which item1 occurs, in which item2 also occurs, calculated with this formula: $cntb / cnt1$ For example, if, in the same store, the number of times that a customer buys both milk (item1) and butter (item2) is 3 (cntb) and the number of times that a customer buys milk is 4 (cnt1), then the confidence that a person who buys milk will also buy butter is $3/4 = 0.75 = 75\%$ .
lift	DOUBLE PRECISION	Ratio of observed support value to expected support value if item1 and item2 were independent; that is: $support(\text{item1 and item2}) / [support(\text{item1}) * support(\text{item2})]$ Value is calculated with this formula: $(cntb / tran\_cnt) / [(cnt1 / tran\_cnt) * (cnt2 / tran\_cnt)]$ If Lift > 1, the occurrence of item1 or item2 has a positive effect on the occurrence of the other items.

Column	Data Type	Description
		If Lift = 1, the occurrence of item1 or item2 has a no effect on the occurrence of the other items. If Lift < 1, the occurrence of item1 or item2 has a negative effect on the occurrence of the other items.
z_score	DOUBLE PRECISION	Significance of co-occurrence, assuming that cntb follows a normal distribution, calculated with this formula: $(cntb - \text{mean}(cntb)) / \text{sd}(cntb)$ If all <i>cntb</i> values are equal, then <i>sd(cntb)</i> is 0, and function does not calculate zscore.

## Deleting Duplicate Output Table Rows

Duplicate output table rows appear because each pair of items appears in two rows—one row has item1 in col1\_item1 and item2 in col1\_item2, and the other row has item2 in col1\_item1 and item1 in col1\_item2. To delete duplicate output table rows, use this code (where *output\_table* is the output table name):

```
DROP TABLE copy;

CREATE MULTISET TABLE copy AS (
  SELECT *, ROW_NUMBER() OVER(ORDER BY col1_item1, col1_item2) rn
  FROM output_table
) WITH DATA;

DROP TABLE DuplicatesRemoved;

CREATE MULTISET TABLE DuplicatesRemoved AS (
  SELECT * FROM copy
) WITH DATA;

DELETE FROM DuplicatesRemoved WHERE rn IN (
  SELECT a.rn FROM DuplicatesRemoved a
  JOIN copy b
  ON a.col1_item1=b.col1_item2 AND a.col1_item2=b.col1_item1 AND a.rn < b.rn
);

DROP TABLE copy;
```

### CFilter DuplicatesRemoved Table Schema

Column	Data Type	Description
col1_item1	VARCHAR	Name of item1.
col1_item2	VARCHAR	Name of item2.

Column	Data Type	Description
rn	INTEGER	Row number in <i>output_table</i> when ordered by col1_item1, col1_item2.

## CFilter Examples

### CFilter Example: Filter by Product

Collaborative filtering by product is also called item-based collaborative filtering. In this example, JoinColumns = 'orderid'. The function tries to identify products that are often bought in the same transaction (as identified by the order\_id).

#### Input

The InputTable has sales transaction data from an office supply chain store, in these columns:

Column	Description
orderid	Order (transaction) identifier
orderdate	Order date
orderqty	Quantity of product ordered
region	Geographic region of store where order was placed
customer_segment	Segment of customer who ordered product
prd_category	Category of product ordered
product	Product ordered

**InputTable: sales\_transaction**

orderid	orderdate	orderqty	region	customer_segment	prd_category	product
3	2010-10-13 00:00:00	6	Nunavut	Small Business	Office Supplies	Storage & Organization
293	2012-10-01 00:00:00	49	Nunavut	Consumer	Office Supplies	Appliances
293	2012-10-01 00:00:00	27	Nunavut	Consumer	Office Supplies	Binders and Binder Accessories
483	2011-07-10 00:00:00	30	Nunavut	Corporate	Technology	Telephones and Communication
515	2010-08-28 00:00:00	19	Nunavut	Consumer	Office Supplies	Appliances

orderid	orderdate	orderqty	region	customer_segment	prd_category	product
515	2010-08-28 00:00:00	21	Nunavut	Consumer	Furniture	Office Furnishings
613	2011-06-17 00:00:00	12	Nunavut	Corporate	Office Supplies	Binders and Binder Accessories
613	2011-06-17 00:00:00	22	Nunavut	Corporate	Office Supplies	Storage & Organization
643	2011-03-24 00:00:00	21	Nunavut	Corporate	Office Supplies	Storage & Organization
678	2010-02-26 00:00:00	44	Nunavut	Home Office	Office Supplies	Paper
807	2010-11-23 00:00:00	45	Nunavut	Home Office	Office Supplies	Paper
807	2010-11-23 00:00:00	32	Nunavut	Home Office	Office Supplies	Rubber Bands
868	2012-06-08 00:00:00	32	Nunavut	Home Office	Office Supplies	Appliances
...	...	...	...	...	...	...

## SQL Call

```
SELECT * FROM CFilter (
  ON sales_transaction AS InputTable
  OUT TABLE OutputTable (cfilter_output)
  USING
    TargetColumns ('product')
    JoinColumns ('orderid')
    PartitionColumns ('region')
) AS dt;
```

## Output

message

```
-----
Output table created successfully
(1 row)
```

```
SELECT * FROM cfilter_output;
```

region	coll_item1	coll_item2
cntb cnt1 cnt2 score	support	confidence

```

lift          z_score
-----
-----
-----
-----
northwest territories binders and binder accessories
labels          2  44  16  0.005681818181818182
0.007434944237918215 0.045454545454545456 0.7642045454545454 0.12784297268860556
northwest territories tables          labels
1  12  16  0.005208333333333333 0.0037174721189591076 0.08333333333333333
1.4010416666666667 -0.8522864845907044
northwest territories computer peripherals          labels
1  30  16  0.002083333333333333 0.0037174721189591076 0.03333333333333333
0.5604166666666667 -0.8522864845907044
nunavut          tables          bookcases
1  1  2          0.5  0.017857142857142856
1.0          28.0 -0.4588314677411239
northwest territories binders and binder accessories
tables          2  44  12  0.007575757575757576
0.007434944237918215 0.045454545454545456 1.018939393939394 0.12784297268860556
nunavut          labels          rubber bands
1  4  2          0.125  0.017857142857142856
0.25          7.0 -0.4588314677411239
nunavut          binders and binder accessories computer
peripherals          1  10  6  0.016666666666666666
0.017857142857142856          0.1  0.9333333333333333 -0.4588314677411239
northwest territories labels          binders and binder
accessories  2  16  44  0.005681818181818182 0.007434944237918215
0.125  0.7642045454545454 0.12784297268860556
northwest territories binders and binder accessories rubber
bands          3  44  10  0.020454545454545454 0.011152416356877323
0.06818181818181818 1.834090909090909 1.1079724299679155
northwest territories labels          paper
2  16  40          0.00625 0.007434944237918215          0.125
0.840625 0.12784297268860556
northwest territories computer peripherals          binders and binder
accessories  3  30  44  0.006818181818181818
0.011152416356877323          0.1  0.6113636363636363 1.1079724299679155
atlantic          labels          pens & art supplies
1  3  2  0.16666666666666666 0.05555555555555555
0.3333333333333333          3.0          NULL
...

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## CFilter Example: Filter by Customer Segment

In this example, JoinColumns = 'product'. The function tries to identify segments of customers that often purchase the same products.

### Input

- InputTable: sales\_transaction, as in [CFilter Example: Filter by Product](#)

### SQL Call

```
SELECT * FROM CFilter (
  ON sales_transaction AS InputTable
  OUT TABLE OutputTable (cfilter_output1)
  USING
  TargetColumns ('customer_segment')
  JoinColumns ('product')
) AS dt;
```

### Output

message

```
-----
Output table created successfully
(1 row)
```

```
SELECT * FROM cfilter_output1;
```

coll_item1	coll_item2	cntb	cnt1	cnt2	score	support
confidence	lift	z_score				
corporate	small business	17	17	17		1.0
1.0	1.0	1.0	1.3728129459672886			
home office	small business	16	16	17	0.9411764705882353	
0.9411764705882353		1.0	1.0	0.7844645405527365		
consumer	small business	13	13	17	0.7647058823529411	
0.7647058823529411		1.0	1.0	-0.9805806756909198		
home office	consumer	13	16	13		0.8125
0.7647058823529411		0.8125	1.0625	-0.9805806756909198		
consumer	corporate	13	13	17	0.7647058823529411	
0.7647058823529411		1.0	1.0	-0.9805806756909198		
small business	corporate	17	17	17		1.0
1.0	1.0	1.0	1.3728129459672886			

```

corporate      home office      16  17  16 0.9411764705882353 0.9411764705882353
0.9411764705882353      1.0  0.7844645405527365
small business home office      16  17  16 0.9411764705882353 0.9411764705882353
0.9411764705882353      1.0  0.7844645405527365
corporate      consumer         13  17  13 0.7647058823529411 0.7647058823529411
0.7647058823529411      1.0 -0.9805806756909198
small business consumer         13  17  13 0.7647058823529411 0.7647058823529411
0.7647058823529411      1.0 -0.9805806756909198
consumer        home office      13  13  16                0.8125
0.7647058823529411                1.0 1.0625 -0.9805806756909198
home office     corporate         16  16  17 0.9411764705882353
0.9411764705882353                1.0    1.0  0.7844645405527365

```

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

# 5.0 Validation and Evaluation

# Receiver Operating Characteristic

**Source:** [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

A **receiver operating characteristic curve**, or **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or *probability of detection* in machine learning. The false-positive rate is also known as *probability of false alarm* and can be calculated as  $(1 - \text{specificity})$ . It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity or recall as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from  $-\infty$  to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis.

ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.

The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields and was soon introduced to psychology to account for perceptual detection of stimuli. ROC analysis since then has been used in medicine, radiology, biometrics, forecasting of natural hazards, meteorology, model performance assessment, and other areas for many decades and is increasingly used in machine learning and data mining research.

The ROC is also known as a relative operating characteristic curve, because it is a comparison of two operating characteristics (TPR and FPR) as the criterion changes.

A classification model (classifier or diagnosis) is a mapping of instances between certain classes/groups. Because the classifier or diagnosis result can be an arbitrary real value (continuous output), the classifier boundary between classes must be determined by a threshold value (for instance, to determine whether a person has hypertension based on a blood pressure measure). Or it can be a discrete class label, indicating one of the classes.

Consider a two-class prediction problem (binary classification), in which the outcomes are labeled either as positive ( $p$ ) or negative ( $n$ ). There are four possible outcomes from a binary classifier. If the outcome from a prediction is  $p$  and the actual value is also  $p$ , then it is called a *true positive* (TP); however if the actual value is  $n$  then it is said to be a *false positive* (FP). Conversely, a *true negative* (TN) has occurred when both the prediction outcome and the actual value are  $n$ , and *false negative* (FN) is when the prediction outcome is  $n$  while the actual value is  $p$ .

To get an appropriate example in a real-world problem, consider a diagnostic test that seeks to determine whether a person has a certain disease. A false positive in this case occurs when the person tests positive, but does not actually have the disease. A false negative, on the other hand, occurs when the person tests negative, suggesting they are healthy, when they actually do have the disease.

**condition positive (P)**

the number of real positive cases in the data

**condition negative (N)**

the number of real negative cases in the data

**sensitivity, recall, hit rate, or true positive rate (TPR)**

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR$$

**specificity, selectivity or true negative rate (TNR)**

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR$$

**precision or positive predictive value (PPV)**

$$PPV = \frac{TP}{TP + FP} = 1 - FDR$$

**negative predictive value (NPV)**

$$NPV = \frac{TN}{TN + FN} = 1 - FOR$$

**miss rate or false negative rate (FNR)**

$$FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR$$

**fall-out or false positive rate (FPR)**

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR$$

**false discovery rate (FDR)**

$$FDR = \frac{FP}{FP + TP} = 1 - PPV$$

**false omission rate (FOR)**

$$FOR = \frac{FN}{FN + TN} = 1 - NPV$$

**Prevalence Threshold (PT)**

$$PT = \frac{\sqrt{TPR(-TNR+1)} + TNR - 1}{(TPR + TNR - 1)}$$

**Threat score (TS) or critical success index (CSI)**

$$TS = \frac{TP}{TP + FN + FP}$$

**true positive (TP)**

eqv. with hit

**true negative (TN)**

eqv. with correct rejection

**false positive (FP)**

eqv. with false alarm, Type I error

**false negative (FN)**

eqv. with miss, Type II error

**accuracy (ACC)**

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

**balanced accuracy (BA)**

$$BA = \frac{TPR + TNR}{2}$$

**F1 score**

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$$

**Matthews correlation coefficient (MCC)**

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

**Fowlkes–Mallows index (FM)**

$$FM = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} = \sqrt{PPV \cdot TPR}$$

**informedness or bookmaker informedness (BM)**

$$BM = TPR + TNR - 1$$

**markedness (MK) or deltaP**

$$MK = PPV + NPV - 1$$

		True condition			
Total population		Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{Condition positive}}{\Sigma \text{Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{True positive} + \Sigma \text{True negative}}{\Sigma \text{Total population}}$
Predicted condition	Predicted condition positive	<b>True positive</b>	<b>False positive, Type I error</b>	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{True positive}}{\Sigma \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{False positive}}{\Sigma \text{Predicted condition positive}}$
	Predicted condition negative	<b>False negative, Type II error</b>	<b>True negative</b>	False omission rate (FOR) = $\frac{\Sigma \text{False negative}}{\Sigma \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{True positive}}{\Sigma \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{False positive}}{\Sigma \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$  F <sub>1</sub> score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{False negative}}{\Sigma \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{FNR}{TNR}$	

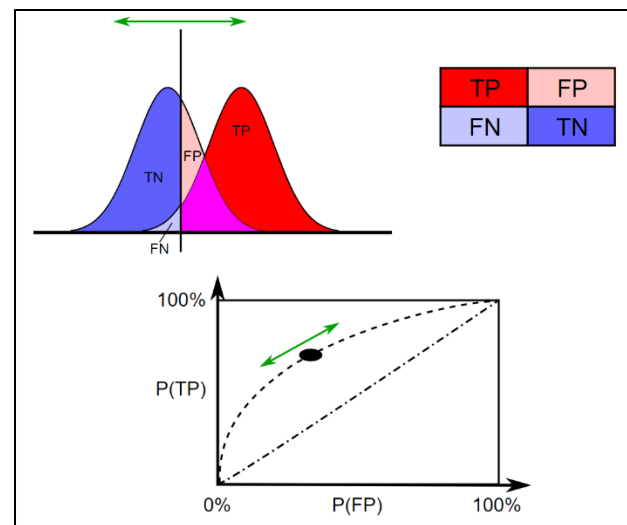
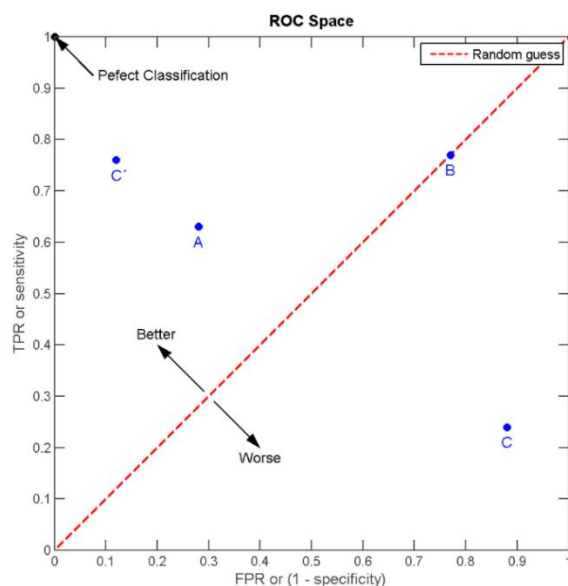
## ROC space

The contingency table can derive several evaluation "metrics" (see infobox). To draw an ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

An ROC space is defined by FPR and TPR as x and y axes, respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since TPR is equivalent to sensitivity and FPR is equal to  $1 - \text{specificity}$ , the ROC graph is sometimes called the sensitivity vs  $(1 - \text{specificity})$  plot. Each prediction result or instance of a confusion matrix represents one point in the ROC space.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a *perfect classification*. A random guess would give a point along a diagonal line (the so-called *line of no-discrimination*) from the left bottom to the top right corners (regardless of the positive and negative base rates). An intuitive example of random guessing is a decision by flipping coins. As the size of the sample increases, a random classifier's ROC point tends towards the diagonal line. In the case of a balanced coin, it will tend to the point (0.5, 0.5).

The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random); points below the line represent bad results (worse than random). Note that the output of a consistently bad predictor could simply be inverted to obtain a good predictor.



Let us look into four prediction results from 100 positive and 100 negative instances:

A			B			C			C'		
TP=63	FP=28	91	TP=77	FP=77	154	TP=24	FP=88	112	TP=76	FP=12	88
FN=37	TN=72	109	FN=23	TN=23	46	FN=76	TN=12	88	FN=24	TN=88	112
100	100	200	100	100	200	100	100	200	100	100	200
TPR = 0.63			TPR = 0.77			TPR = 0.24			TPR = 0.76		
FPR = 0.28			FPR = 0.77			FPR = 0.88			FPR = 0.12		
PPV = 0.69			PPV = 0.50			PPV = 0.21			PPV = 0.86		
F1 = 0.66			F1 = 0.61			F1 = 0.23			F1 = 0.81		
ACC = 0.68			ACC = 0.50			ACC = 0.18			ACC = 0.82		

Plots of the four results above in the ROC space are given in the figure. The result of method **A** clearly shows the best predictive power among **A**, **B**, and **C**. The result of **B** lies on the random guess line (the diagonal line), and it can be seen in the table that the accuracy of **B** is 50%. However, when **C** is mirrored across the center point (0.5,0.5), the resulting method **C'** is even better than **A**. This mirrored method simply reverses the predictions of whatever method or test produced the **C** contingency table. Although the original **C** method has negative predictive power, simply reversing its decisions leads to a new predictive method **C'** which has positive predictive power. When the **C** method predicts **p** or **n**, the **C'** method would predict **n** or **p**, respectively. In this manner, the **C'** test would perform the best. The closer a result from a contingency table is to the upper left corner, the better it predicts, but the distance from the random guess line in either direction is the best indicator of how much predictive power a method has. If the result is below the line (i.e. the method is worse than a random guess), all of the method's predictions must be reversed in order to utilize its power, thereby moving the result above the random guess line.

Sometimes, the ROC is used to generate a summary statistic. Common versions are:

- the intercept of the ROC curve with the line at 45 degrees orthogonal to the no-discrimination line - the balance point where Sensitivity = 1 - Specificity
- the intercept of the ROC curve with the tangent at 45 degrees parallel to the no-discrimination line that is closest to the error-free point (0,1) - also called Youden's J statistic and generalized as Informedness
- the area between the ROC curve and the no-discrimination line multiplied by two - Gini Coefficient
- the area between the full ROC curve and the triangular ROC curve including only (0,0), (1,1) and one selected operating point (tpr,fpr) - Consistency
- the area under the ROC curve, or "AUC" ("Area Under Curve"), or A' (pronounced "a-prime"), or "c-statistic" ("concordance statistic").
- the sensitivity index  $d'$  (pronounced "d-prime"), the distance between the mean of the distribution of activity in the system under noise-alone conditions and its distribution under signal-alone conditions, divided by their standard deviation, under the assumption that both these distributions are normal with the same standard deviation. Under these assumptions, the shape of the ROC is entirely determined by  $d'$ .

# Receiver Operating Characteristic (ROC) (ML Engine)

A receiver operating characteristic (ROC) curve shows the performance of a binary classification model as its discrimination threshold varies. For a range of thresholds, the curve plots the true positive rate against the false positive rate.

The Receiver Operating Characteristic (ROC) function takes a set of prediction-actual pairs for a binary classification model and calculates the following values for a range of discrimination thresholds:

- True positive rate (TPR)
- False positive rate (FPR)
- Area under the ROC curve (AUC)
- Gini coefficient

A prediction-actual pair for a binary classifier consists of:

- Predicted probability that an observation is in the positive class
- Actual class of the observation

A discrimination threshold determines whether an observation is classified as positive (1) or negative (0). For example, suppose that a model predicts that an observation will be classified as positive with 0.55 probability. If the threshold above which an observation is classified as positive is 0.5, then the observation is classified as positive. If the threshold is 0.6, the observation is classified as negative.

You can create prediction-actual pairs for ROC with these functions:

- [AdaBoostPredict](#)
- [DecisionForestPredict\\_MLE](#)
- [DecisionTreePredict\\_MLE](#)
- [GLMPredict\\_MLE](#)
- [XGBoostPredict](#)

## ROC Syntax

### Version 1.8

```
SELECT * FROM ROC (  
  ON { table | view | (query) } AS InputTable  
  { OUT TABLE OutputTable (output_table) |  
    OUT TABLE ROCTable (ROC_table) }  
  USING  
  [ ModelIDColumn ('model_id_column') ]  
  ProbabilityColumn ('probability_column')  
  ObservationColumn ('observation_column')  
  PositiveClass ('positive_class_label')
```

```
[ NumThresholds (num_thresholds)]
[ ROCValues ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ AUC ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ Gini ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
) AS alias;
```

## ROC Syntax Elements

### OutputTable

[Required if you omit ROCTable, disallowed otherwise.] Specify the name for the output table that the function creates. The *output\_table* must not already exist.

### ROCTable

[Required if you omit OutputTable, disallowed otherwise.] Specify the name for the ROC table that the function creates. The *ROC\_table* must not already exist.

### ModelIDColumn

[Optional] Specify the name of the InputTable column that contains the model or partition identifiers for the ROC curves.

Use this syntax element only when InputTable contains information for more than one model. The function creates a separate ROC curve for each model identifier in this column. Each model must include exactly two classes in ObservationColumn.

### ProbabilityColumn

Specify the name of the InputTable column that contains the predictions.

### ObservationColumn

Specify the name of the InputTable column that contains the actual classes.

### PositiveClass

Specify the label of the positive class.

### NumThresholds

[Optional] Specify the number of thresholds for the function to use. The *num\_thresholds* must be a NUMERIC value in the range [1, 10000].

Default: 50 (The function uniformly distributes the thresholds between 0 and 1.)

### ROCValues

[Optional with OutputTable, disallowed with ROCTable.] Specify whether the function displays ROC values (thresholds, false positive rates, and true positive rates).

Default: 'true'. See the following note.

### AUC

[Optional with OutputTable, disallowed with ROCTable.] Specify whether the function displays the AUC calculated from the ROC values.

Default: 'false'. See the following note.

## Gini

[Optional with OutputTable, disallowed with ROCTable.] Specify whether the function displays the Gini coefficient calculated from the ROC values. The Gini coefficient is a measure of inequality among values of a frequency distribution. A Gini coefficient of 0 indicates that all values are the same. The closer the Gini coefficient is to 1, the more unequal are the values in the distribution.

Default: 'false'. See the following note.

If you specify OutputTable, the valid combinations of ROCValues, AUC, and Gini syntax elements are those that specify one of the following:

- ROCValues only
- AUC only
- Gini only
- AUC and Gini

The function issues an error message if you do any of the following:

- Specify AUC only, Gini only, or AUC and Gini only, and ROCValues ('true').  
(When specifying AUC only, Gini only, or AUC and Gini only, ROCValues is false by default.)
- Specify an invalid combination (such as ROCValues ('true') and AUC ('true'), or all three 'false').
- Specify ROCTable and also specify any of AUC, Gini, or ROCValues.

## ROC Input

### Input Table Schema

The table has one row for each observation.

Column	Data Type	Description
<i>model_id_column</i>	Any	[Column appears only with ModelIDColumn syntax element.] Model identifier or partition for ROC curve associated with observation.
<i>probability_column</i>	DOUBLE PRECISION	Predicted probability that observation is in positive class.
<i>observation_column</i>	Any	Actual class of observation.

## ROC Output

The output depends on whether you specify OutputTable or ROCTable.

## ROC Output with OutputTable

If you specify OutputTable, the function outputs a message and creates OutputTable. The OutputTable schema depends on the ROCValues syntax element.

### Output Message Schema

Column	Data Type	Description
info	VARCHAR	Reports whether function completed.

### OutputTable Schema, ROCValues ('true') (Default)

The table has one row for each threshold for each model, and contains only ROC values.

Column	Data Type	Description
model	Same as <i>model_id_column</i> in input table	[Column appears only with ModelIDColumn syntax element.] Model identifier or partition for ROC curve associated with observation, taken from <i>model_id_column</i> .
threshold	DOUBLE PRECISION	Threshold at which function classifies an observation as positive.
tpr	DOUBLE PRECISION	True positive rate for threshold (number of observations correctly predicted as positive based on threshold, divided by number of observations known to be positive).
fpr	DOUBLE PRECISION	False positive rate for threshold (number of observations incorrectly predicted as positive based on threshold, divided by number of observations known to be negative).

### OutputTable Schema, ROCValues ('false')

This is the default output table if you specify AUC only, Gini only, or AUC and Gini only.

The table has the following:

- One row for each model
- No ROC values
- AUC values, Gini values, or both (depending on AUC and Gini syntax elements)

Column	Data Type	Description
model	Same as <i>model_id_column</i> in input table	[Column appears only with ModelIDColumn syntax element.] Model identifier or partition for ROC curve associated with observation, taken from <i>model_id_column</i> .
AUC	DOUBLE PRECISION	Area under ROC curve for data in partition. With AUC ('false'), this value is NULL.

Column	Data Type	Description
Gini	DOUBLE PRECISION	Gini coefficient for ROC curve for data in partition. With Gini ('false'), this value is NULL.

## ROC Output with ROCTable

If you specify ROCTable, the function outputs a table to the screen and creates ROCTable.

### Onscreen Output Table Schema

Column	Data Type	Description
model	Same as <i>model_id_column</i> in input table	[Column appears only with ModelIDColumn syntax element.] Model identifier or partition for ROC curve associated with observation, taken from <i>model_id_column</i> .
auc	DOUBLE PRECISION	Area under ROC curve for data in partition.
gini	DOUBLE PRECISION	Gini coefficient for ROC curve for data in partition.

### ROCTable Schema

Same as default OutputTable Schema in [ROC Output with OutputTable](#).

## ROC Examples

### ROC Example: OutputTable, Default Values

#### Input

All ROC examples use this input table, roc\_input, which has data from four different models:

```

model_id id  observation  probability
-----
      1   7   0           0.15
      1  40   0           0.4
      1  55   1           1.0
      1  57   1           0.85
      1  72   1           1.0
      1  80   1           0.9
      1  95   1           1.0
      1 110   0           0.0
      1 112   0           0.0
      1 118   0           0.0
      1 120   1           0.9

```

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

1 127 0	0.0
1 135 0	0.0
1 150 0	0.1
1 158 1	1.0
1 162 0	0.05
1 167 0	0.0
1 173 0	0.0
1 175 0	0.0
1 190 1	0.9
1 200 0	0.0
1 213 1	0.1
1 217 0	0.0
1 223 1	1.0
1 228 0	0.0
2 230 0	0.05
2 240 0	0.0
2 255 0	0.0
2 268 1	1.0
2 270 1	1.0
2 272 1	1.0
2 278 1	0.9
2 295 1	1.0
2 310 1	0.9
2 343 0	0.0
2 345 0	0.05
2 360 0	0.15
2 383 1	1.0
2 398 0	0.3
2 400 1	0.95
2 406 0	0.0
2 415 0	0.0
2 423 0	0.0
2 438 0	0.0
2 446 1	1.0
2 453 0	0.0
2 455 0	0.05
2 461 0	0.1
2 463 0	0.0
2 478 1	0.85
3 488 1	0.75
3 493 1	1.0
3 501 1	1.0
3 503 0	0.4
3 505 0	0.35

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

3 516 0	0.0
3 518 0	0.0
3 528 0	0.15
3 533 1	0.9
3 543 1	0.7
3 556 1	0.7
3 558 1	1.0
3 560 1	0.95
3 575 1	0.35
3 583 0	0.0
3 598 1	1.0
3 615 1	0.45
3 631 1	1.0
3 648 1	0.9
3 671 0	0.05
3 686 0	0.0
3 688 0	0.1
3 703 0	0.0
3 711 0	0.8
3 718 0	0.15
4 726 1	1.0
4 734 0	0.05
4 741 1	0.4
4 743 0	0.05
4 749 0	0.0
4 751 0	0.0
4 758 0	0.0
4 766 1	0.85
4 781 0	0.7
4 789 0	0.1
4 791 1	0.7
4 793 1	1.0
4 798 0	0.0
4 804 1	1.0
4 806 0	0.0
4 808 1	0.9
4 821 1	1.0
4 831 0	0.0
4 846 0	0.0
4 848 0	0.0
4 861 1	0.8
4 863 0	0.05
4 886 0	0.3

4 901 0	0.0
4 903 0	0.0

## SQL Call

In this call, the ROCValues, AUC, and Gini syntax elements default to the values 'true', 'false', and 'false', respectively.

```
SELECT * FROM ROC (
  ON roc_input AS InputTable
  OUT TABLE OutputTable (roc_out_1)
  USING
    ModelIdColumn ('model_id')
    ProbabilityColumn ('probability')
    ObservationColumn ('observation')
    PositiveClass ('1')
    NumThresholds (100)
) AS dt;
```

## Output

```
info
-----
ROC complete.
```

```
SELECT * FROM roc_out_1;
```

model_id	threshold	tpr	fpr
1	0.0	1.0	1.0
1	0.0101010101010101	1.0	0.266666666666667
1	0.0202020202020202	1.0	0.266666666666667
1	0.0303030303030303	1.0	0.266666666666667
1	0.0404040404040404	1.0	0.266666666666667
1	0.0505050505050505	1.0	0.2
1	0.0606060606060606	1.0	0.2
1	0.0707070707070707	1.0	0.2
1	0.0808080808080808	1.0	0.2
1	0.0909090909090909	1.0	0.2
1	0.1010101010101010	0.9	0.133333333333333
1	0.1111111111111111	0.9	0.133333333333333
1	0.1212121212121212	0.9	0.133333333333333
1	0.1313131313131313	0.9	0.133333333333333
1	0.1414141414141414	0.9	0.133333333333333
1	0.1515151515151515	0.9	0.066666666666667
1	0.1616161616161616	0.9	0.066666666666667

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

1	0.17171717171717172	0.9	0.06666666666666667
1	0.18181818181818182	0.9	0.06666666666666667
1	0.19191919191919192	0.9	0.06666666666666667
1	0.20202020202020202	0.9	0.06666666666666667
1	0.21212121212121212	0.9	0.06666666666666667
1	0.22222222222222222	0.9	0.06666666666666667
1	0.23232323232323232	0.9	0.06666666666666667
1	0.24242424242424242	0.9	0.06666666666666667
1	0.25252525252525253	0.9	0.06666666666666667
1	0.26262626262626263	0.9	0.06666666666666667
1	0.27272727272727273	0.9	0.06666666666666667
1	0.28282828282828283	0.9	0.06666666666666667
1	0.29292929292929293	0.9	0.06666666666666667
1	0.30303030303030303	0.9	0.06666666666666667
1	0.31313131313131313	0.9	0.06666666666666667
1	0.32323232323232323	0.9	0.06666666666666667
1	0.33333333333333333	0.9	0.06666666666666667
1	0.34343434343434343	0.9	0.06666666666666667
1	0.35353535353535354	0.9	0.06666666666666667
1	0.36363636363636364	0.9	0.06666666666666667
1	0.37373737373737374	0.9	0.06666666666666667
1	0.38383838383838384	0.9	0.06666666666666667
1	0.39393939393939394	0.9	0.06666666666666667
1	0.40404040404040404	0.9	0.0
1	0.41414141414141414	0.9	0.0
1	0.42424242424242424	0.9	0.0
1	0.43434343434343434	0.9	0.0
1	0.44444444444444444	0.9	0.0
1	0.45454545454545455	0.9	0.0
1	0.46464646464646465	0.9	0.0
1	0.47474747474747475	0.9	0.0
1	0.48484848484848485	0.9	0.0
1	0.49494949494949495	0.9	0.0
1	0.50505050505050505	0.9	0.0
1	0.51515151515151515	0.9	0.0
1	0.52525252525252525	0.9	0.0
1	0.53535353535353535	0.9	0.0
1	0.54545454545454546	0.9	0.0
1	0.55555555555555556	0.9	0.0
1	0.56565656565656566	0.9	0.0
1	0.57575757575757576	0.9	0.0
1	0.58585858585858586	0.9	0.0
1	0.59595959595959596	0.9	0.0
1	0.60606060606060606	0.9	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

1	0.616161616161616	0.9	0.0
1	0.626262626262626	0.9	0.0
1	0.636363636363636	0.9	0.0
1	0.646464646464647	0.9	0.0
1	0.656565656565657	0.9	0.0
1	0.666666666666667	0.9	0.0
1	0.676767676767677	0.9	0.0
1	0.686868686868687	0.9	0.0
1	0.696969696969697	0.9	0.0
1	0.707070707070707	0.9	0.0
1	0.717171717171717	0.9	0.0
1	0.727272727272727	0.9	0.0
1	0.737373737373737	0.9	0.0
1	0.747474747474748	0.9	0.0
1	0.757575757575758	0.9	0.0
1	0.767676767676768	0.9	0.0
1	0.777777777777778	0.9	0.0
1	0.787878787878788	0.9	0.0
1	0.797979797979798	0.9	0.0
1	0.808080808080808	0.9	0.0
1	0.818181818181818	0.9	0.0
1	0.828282828282828	0.9	0.0
1	0.838383838383838	0.9	0.0
1	0.848484848484849	0.9	0.0
1	0.858585858585859	0.8	0.0
1	0.868686868686869	0.8	0.0
1	0.878787878787879	0.8	0.0
1	0.888888888888889	0.8	0.0
1	0.898989898989899	0.8	0.0
1	0.909090909090909	0.5	0.0
1	0.919191919191919	0.5	0.0
1	0.929292929292929	0.5	0.0
1	0.939393939393939	0.5	0.0
1	0.94949494949495	0.5	0.0
1	0.95959595959596	0.5	0.0
1	0.96969696969697	0.5	0.0
1	0.97979797979798	0.5	0.0
1	0.98989898989899	0.5	0.0
1	1.0	0.5	0.0
2	0.0	1.0	1.0
2	0.010101010101010	1.0	0.4
2	0.020202020202020	1.0	0.4
2	0.030303030303030	1.0	0.4
2	0.040404040404040	1.0	0.4

# 13: Receiver Operating Characteristic (ROC) (ML Engine)

2 0.0505050505050505	1.0	0.2
2 0.0606060606060606	1.0	0.2
2 0.0707070707070707	1.0	0.2
2 0.0808080808080808	1.0	0.2
2 0.0909090909090909	1.0	0.2
2 0.1010101010101010	1.0	0.1333333333333333
2 0.1111111111111111	1.0	0.1333333333333333
2 0.1212121212121212	1.0	0.1333333333333333
2 0.1313131313131313	1.0	0.1333333333333333
2 0.1414141414141414	1.0	0.1333333333333333
2 0.1515151515151515	1.0	0.0666666666666667
2 0.1616161616161616	1.0	0.0666666666666667
2 0.1717171717171717	1.0	0.0666666666666667
2 0.1818181818181818	1.0	0.0666666666666667
2 0.1919191919191919	1.0	0.0666666666666667
2 0.2020202020202020	1.0	0.0666666666666667
2 0.2121212121212121	1.0	0.0666666666666667
2 0.2222222222222222	1.0	0.0666666666666667
2 0.2323232323232323	1.0	0.0666666666666667
2 0.2424242424242424	1.0	0.0666666666666667
2 0.2525252525252525	1.0	0.0666666666666667
2 0.2626262626262626	1.0	0.0666666666666667
2 0.2727272727272727	1.0	0.0666666666666667
2 0.2828282828282828	1.0	0.0666666666666667
2 0.2929292929292929	1.0	0.0666666666666667
2 0.3030303030303030	1.0	0.0
2 0.3131313131313131	1.0	0.0
2 0.3232323232323232	1.0	0.0
2 0.3333333333333333	1.0	0.0
2 0.3434343434343434	1.0	0.0
2 0.3535353535353535	1.0	0.0
2 0.3636363636363636	1.0	0.0
2 0.3737373737373737	1.0	0.0
2 0.3838383838383838	1.0	0.0
2 0.3939393939393939	1.0	0.0
2 0.4040404040404040	1.0	0.0
2 0.4141414141414141	1.0	0.0
2 0.4242424242424242	1.0	0.0
2 0.4343434343434343	1.0	0.0
2 0.4444444444444444	1.0	0.0
2 0.4545454545454545	1.0	0.0
2 0.4646464646464646	1.0	0.0
2 0.4747474747474747	1.0	0.0
2 0.4848484848484848	1.0	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

2	0.494949494949495	1.0	0.0
2	0.505050505050505	1.0	0.0
2	0.515151515151515	1.0	0.0
2	0.525252525252525	1.0	0.0
2	0.535353535353535	1.0	0.0
2	0.545454545454546	1.0	0.0
2	0.555555555555556	1.0	0.0
2	0.565656565656566	1.0	0.0
2	0.575757575757576	1.0	0.0
2	0.585858585858586	1.0	0.0
2	0.595959595959596	1.0	0.0
2	0.606060606060606	1.0	0.0
2	0.616161616161616	1.0	0.0
2	0.626262626262626	1.0	0.0
2	0.636363636363636	1.0	0.0
2	0.646464646464647	1.0	0.0
2	0.656565656565657	1.0	0.0
2	0.666666666666667	1.0	0.0
2	0.676767676767677	1.0	0.0
2	0.686868686868687	1.0	0.0
2	0.696969696969697	1.0	0.0
2	0.707070707070707	1.0	0.0
2	0.717171717171717	1.0	0.0
2	0.727272727272727	1.0	0.0
2	0.737373737373737	1.0	0.0
2	0.747474747474748	1.0	0.0
2	0.757575757575758	1.0	0.0
2	0.767676767676768	1.0	0.0
2	0.777777777777778	1.0	0.0
2	0.787878787878788	1.0	0.0
2	0.797979797979798	1.0	0.0
2	0.808080808080808	1.0	0.0
2	0.818181818181818	1.0	0.0
2	0.828282828282828	1.0	0.0
2	0.838383838383838	1.0	0.0
2	0.848484848484849	1.0	0.0
2	0.858585858585859	0.9	0.0
2	0.868686868686869	0.9	0.0
2	0.878787878787879	0.9	0.0
2	0.888888888888889	0.9	0.0
2	0.898989898989899	0.9	0.0
2	0.909090909090909	0.7	0.0
2	0.919191919191919	0.7	0.0
2	0.929292929292929	0.7	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

2	0.939393939393939	0.7	0.0
2	0.949494949494949	0.7	0.0
2	0.959595959595959	0.6	0.0
2	0.969696969696969	0.6	0.0
2	0.979797979797979	0.6	0.0
2	0.989898989898989	0.6	0.0
2	1.0	0.6	0.0
3	0.0	1.0	1.0
3	0.010101010101010	1.0	0.583333333333333
3	0.020202020202020	1.0	0.583333333333333
3	0.030303030303030	1.0	0.583333333333333
3	0.040404040404040	1.0	0.583333333333333
3	0.050505050505050	1.0	0.5
3	0.060606060606060	1.0	0.5
3	0.070707070707070	1.0	0.5
3	0.080808080808080	1.0	0.5
3	0.090909090909090	1.0	0.5
3	0.101010101010101	1.0	0.416666666666667
3	0.111111111111111	1.0	0.416666666666667
3	0.121212121212121	1.0	0.416666666666667
3	0.131313131313131	1.0	0.416666666666667
3	0.141414141414141	1.0	0.416666666666667
3	0.151515151515152	1.0	0.25
3	0.161616161616162	1.0	0.25
3	0.171717171717172	1.0	0.25
3	0.181818181818182	1.0	0.25
3	0.191919191919192	1.0	0.25
3	0.202020202020202	1.0	0.25
3	0.212121212121212	1.0	0.25
3	0.222222222222222	1.0	0.25
3	0.232323232323232	1.0	0.25
3	0.242424242424242	1.0	0.25
3	0.252525252525253	1.0	0.25
3	0.262626262626263	1.0	0.25
3	0.272727272727273	1.0	0.25
3	0.282828282828283	1.0	0.25
3	0.292929292929293	1.0	0.25
3	0.303030303030303	1.0	0.25
3	0.313131313131313	1.0	0.25
3	0.323232323232323	1.0	0.25
3	0.333333333333333	1.0	0.25
3	0.343434343434343	1.0	0.25
3	0.353535353535354	0.923076923076923	0.166666666666667
3	0.363636363636364	0.923076923076923	0.166666666666667

```

3 0.3737373737373737 0.923076923076923 0.166666666666667
3 0.3838383838383838 0.923076923076923 0.166666666666667
3 0.3939393939393939 0.923076923076923 0.166666666666667
3 0.4040404040404040 0.923076923076923 0.0833333333333333
3 0.4141414141414141 0.923076923076923 0.0833333333333333
3 0.4242424242424242 0.923076923076923 0.0833333333333333
3 0.4343434343434343 0.923076923076923 0.0833333333333333
3 0.4444444444444444 0.923076923076923 0.0833333333333333
3 0.4545454545454545 0.846153846153846 0.0833333333333333
3 0.4646464646464646 0.846153846153846 0.0833333333333333
3 0.4747474747474747 0.846153846153846 0.0833333333333333
3 0.4848484848484848 0.846153846153846 0.0833333333333333
3 0.4949494949494949 0.846153846153846 0.0833333333333333
3 0.5050505050505050 0.846153846153846 0.0833333333333333
3 0.5151515151515151 0.846153846153846 0.0833333333333333
3 0.5252525252525252 0.846153846153846 0.0833333333333333
3 0.5353535353535353 0.846153846153846 0.0833333333333333
3 0.5454545454545454 0.846153846153846 0.0833333333333333
3 0.5555555555555555 0.846153846153846 0.0833333333333333
3 0.5656565656565656 0.846153846153846 0.0833333333333333
3 0.5757575757575757 0.846153846153846 0.0833333333333333
3 0.5858585858585858 0.846153846153846 0.0833333333333333
3 0.5959595959595959 0.846153846153846 0.0833333333333333
3 0.6060606060606060 0.846153846153846 0.0833333333333333
3 0.6161616161616161 0.846153846153846 0.0833333333333333
3 0.6262626262626262 0.846153846153846 0.0833333333333333
3 0.6363636363636363 0.846153846153846 0.0833333333333333
3 0.6464646464646464 0.846153846153846 0.0833333333333333
3 0.6565656565656565 0.846153846153846 0.0833333333333333
3 0.6666666666666667 0.846153846153846 0.0833333333333333
3 0.6767676767676767 0.846153846153846 0.0833333333333333
3 0.6868686868686868 0.846153846153846 0.0833333333333333
3 0.6969696969696969 0.846153846153846 0.0833333333333333
3 0.7070707070707070 0.692307692307692 0.0833333333333333
3 0.7171717171717171 0.692307692307692 0.0833333333333333
3 0.7272727272727272 0.692307692307692 0.0833333333333333
3 0.7373737373737373 0.692307692307692 0.0833333333333333
3 0.7474747474747474 0.692307692307692 0.0833333333333333
3 0.7575757575757575 0.615384615384615 0.0833333333333333
3 0.7676767676767676 0.615384615384615 0.0833333333333333
3 0.7777777777777777 0.615384615384615 0.0833333333333333
3 0.7878787878787878 0.615384615384615 0.0833333333333333
3 0.7979797979797979 0.615384615384615 0.0833333333333333
3 0.8080808080808080 0.615384615384615 0.0

```

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

3	0.818181818181818	0.615384615384615	0.0
3	0.828282828282828	0.615384615384615	0.0
3	0.838383838383838	0.615384615384615	0.0
3	0.848484848484849	0.615384615384615	0.0
3	0.858585858585859	0.615384615384615	0.0
3	0.868686868686869	0.615384615384615	0.0
3	0.878787878787879	0.615384615384615	0.0
3	0.888888888888889	0.615384615384615	0.0
3	0.898989898989899	0.615384615384615	0.0
3	0.909090909090909	0.461538461538462	0.0
3	0.919191919191919	0.461538461538462	0.0
3	0.929292929292929	0.461538461538462	0.0
3	0.939393939393939	0.461538461538462	0.0
3	0.94949494949495	0.461538461538462	0.0
3	0.95959595959596	0.384615384615385	0.0
3	0.96969696969697	0.384615384615385	0.0
3	0.97979797979798	0.384615384615385	0.0
3	0.98989898989899	0.384615384615385	0.0
3	1.0	0.384615384615385	0.0
4	0.0	1.0	1.0
4	0.0101010101010101	1.0	0.375
4	0.0202020202020202	1.0	0.375
4	0.0303030303030303	1.0	0.375
4	0.0404040404040404	1.0	0.375
4	0.0505050505050505	1.0	0.1875
4	0.0606060606060606	1.0	0.1875
4	0.0707070707070707	1.0	0.1875
4	0.0808080808080808	1.0	0.1875
4	0.0909090909090909	1.0	0.1875
4	0.10101010101010101	1.0	0.125
4	0.1111111111111111	1.0	0.125
4	0.12121212121212121	1.0	0.125
4	0.13131313131313131	1.0	0.125
4	0.14141414141414141	1.0	0.125
4	0.15151515151515152	1.0	0.125
4	0.16161616161616162	1.0	0.125
4	0.17171717171717172	1.0	0.125
4	0.18181818181818182	1.0	0.125
4	0.19191919191919192	1.0	0.125
4	0.20202020202020202	1.0	0.125
4	0.21212121212121212	1.0	0.125
4	0.22222222222222222	1.0	0.125
4	0.23232323232323232	1.0	0.125
4	0.24242424242424242	1.0	0.125

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

4	0.252525252525253	1.0	0.125
4	0.262626262626263	1.0	0.125
4	0.272727272727273	1.0	0.125
4	0.282828282828283	1.0	0.125
4	0.292929292929293	1.0	0.125
4	0.303030303030303	1.0	0.0625
4	0.313131313131313	1.0	0.0625
4	0.323232323232323	1.0	0.0625
4	0.333333333333333	1.0	0.0625
4	0.343434343434343	1.0	0.0625
4	0.353535353535354	1.0	0.0625
4	0.363636363636364	1.0	0.0625
4	0.373737373737374	1.0	0.0625
4	0.383838383838384	1.0	0.0625
4	0.393939393939394	1.0	0.0625
4	0.404040404040404	0.888888888888889	0.0625
4	0.414141414141414	0.888888888888889	0.0625
4	0.424242424242424	0.888888888888889	0.0625
4	0.434343434343434	0.888888888888889	0.0625
4	0.444444444444444	0.888888888888889	0.0625
4	0.454545454545455	0.888888888888889	0.0625
4	0.464646464646465	0.888888888888889	0.0625
4	0.474747474747475	0.888888888888889	0.0625
4	0.484848484848485	0.888888888888889	0.0625
4	0.494949494949495	0.888888888888889	0.0625
4	0.505050505050505	0.888888888888889	0.0625
4	0.515151515151515	0.888888888888889	0.0625
4	0.525252525252525	0.888888888888889	0.0625
4	0.535353535353535	0.888888888888889	0.0625
4	0.545454545454546	0.888888888888889	0.0625
4	0.555555555555556	0.888888888888889	0.0625
4	0.565656565656566	0.888888888888889	0.0625
4	0.575757575757576	0.888888888888889	0.0625
4	0.585858585858586	0.888888888888889	0.0625
4	0.595959595959596	0.888888888888889	0.0625
4	0.606060606060606	0.888888888888889	0.0625
4	0.616161616161616	0.888888888888889	0.0625
4	0.626262626262626	0.888888888888889	0.0625
4	0.636363636363636	0.888888888888889	0.0625
4	0.646464646464647	0.888888888888889	0.0625
4	0.656565656565657	0.888888888888889	0.0625
4	0.666666666666667	0.888888888888889	0.0625
4	0.676767676767677	0.888888888888889	0.0625
4	0.686868686868687	0.888888888888889	0.0625

4	0.696969696969697	0.888888888888889	0.0625
4	0.707070707070707	0.777777777777778	0.0
4	0.717171717171717	0.777777777777778	0.0
4	0.727272727272727	0.777777777777778	0.0
4	0.737373737373737	0.777777777777778	0.0
4	0.747474747474748	0.777777777777778	0.0
4	0.757575757575758	0.777777777777778	0.0
4	0.767676767676768	0.777777777777778	0.0
4	0.777777777777778	0.777777777777778	0.0
4	0.787878787878788	0.777777777777778	0.0
4	0.797979797979798	0.777777777777778	0.0
4	0.808080808080808	0.666666666666667	0.0
4	0.818181818181818	0.666666666666667	0.0
4	0.828282828282828	0.666666666666667	0.0
4	0.838383838383838	0.666666666666667	0.0
4	0.848484848484849	0.666666666666667	0.0
4	0.858585858585859	0.555555555555556	0.0
4	0.868686868686869	0.555555555555556	0.0
4	0.878787878787879	0.555555555555556	0.0
4	0.888888888888889	0.555555555555556	0.0
4	0.898989898989899	0.555555555555556	0.0
4	0.909090909090909	0.444444444444444	0.0
4	0.919191919191919	0.444444444444444	0.0
4	0.929292929292929	0.444444444444444	0.0
4	0.939393939393939	0.444444444444444	0.0
4	0.94949494949495	0.444444444444444	0.0
4	0.95959595959596	0.444444444444444	0.0
4	0.96969696969697	0.444444444444444	0.0
4	0.97979797979798	0.444444444444444	0.0
4	0.98989898989899	0.444444444444444	0.0
4	1.0	0.444444444444444	0.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## ROC Example: OutputTable, AUC ('true')

This example uses AUC values to check the performance of the model used in [ROC Example: OutputTable, Default Values](#).

### Input

The input table is roc\_input, as in [ROC Example: OutputTable, Default Values](#).

## SQL Call

Because this call specifies AUC ('true') and omits the ROCValues syntax element, the ROCValues syntax element has the value 'false'.

```
SELECT * FROM ROC (
  ON roc_input AS InputTable
  OUT TABLE OutputTable (roc_out_2)
  USING
    ModelIdColumn ('model_id')
    ProbabilityColumn ('probability')
    ObservationColumn ('observation')
    PositiveClass ('1')
    NumThresholds (100)
    AUC ('true')
) AS dt;
```

## Output

```
info
-----
ROC complete.
```

```
SELECT * FROM roc_out_2;
```

model_id	auc	gini
1	0.9833333333333334	NULL
2		1.0 NULL
3	0.9583333333333333	NULL
4	0.9895833333333334	NULL

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## ROC Example: OutputTable, AUC ('true'), Gini ('true')

### Input

The input table is roc\_input, as in [ROC Example: OutputTable, Default Values](#).

### SQL Call

Because this call specifies AUC ('true') and Gini ('true') and omits the ROCValues syntax element, the ROCValues syntax element has the value 'false'.

```
SELECT * FROM ROC (
  ON roc_input AS InputTable
```

```

OUT TABLE OutputTable (roc_out_3)
USING
ProbabilityColumn ('model_id')
ObservationColumn ('observation')
PositiveClass ('1')
NumThresholds (100)
AUC ('true')
Gini ('true')
) AS dt;

```

## Output

```

info
-----
ROC complete.

```

```
SELECT * FROM roc_out_3;
```

model_id	auc	gini
1	0.9833333333333334	0.9666666666666668
2	1.0	1.0
3	0.9583333333333333	0.9166666666666665
4	0.9895833333333334	0.9791666666666667

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

## ROC Example: ROCTable

### Input

The input table is roc\_input, as in [ROC Example: OutputTable, Default Values](#).

### SQL Call

```

SELECT * FROM ROC (
  ON roc_input AS InputTable
  OUT TABLE ROCTable (roc_out_4)
  USING
  ModelIdColumn ('model_id')
  ProbabilityColumn ('probability')
  ObservationColumn ('observation')
  PositiveClass ('1')
  NumThresholds (100)
) AS dt;

```

**Output**

Onscreen:

model_id	auc	gini
1	0.9833333333333334	0.9666666666666668
2		1.0
3	0.9583333333333333	0.9166666666666665
4	0.9895833333333334	0.9791666666666667

The ROCTable, roc\_out\_4, is the same as the OutputTable in [ROC Example: OutputTable, Default Values](#).

```
SELECT * FROM roc_out_4;
```

model_id	threshold	tp	fn
1	0.0	1.0	1.0
1	0.0101010101010101	1.0	0.266666666666667
1	0.0202020202020202	1.0	0.266666666666667
1	0.0303030303030303	1.0	0.266666666666667
1	0.0404040404040404	1.0	0.266666666666667
1	0.0505050505050505	1.0	0.2
1	0.0606060606060606	1.0	0.2
1	0.0707070707070707	1.0	0.2
1	0.0808080808080808	1.0	0.2
1	0.0909090909090909	1.0	0.2
1	0.1010101010101010	0.9	0.133333333333333
1	0.1111111111111111	0.9	0.133333333333333
1	0.1212121212121212	0.9	0.133333333333333
1	0.1313131313131313	0.9	0.133333333333333
1	0.1414141414141414	0.9	0.133333333333333
1	0.1515151515151515	0.9	0.066666666666667
1	0.1616161616161616	0.9	0.066666666666667
1	0.1717171717171717	0.9	0.066666666666667
1	0.1818181818181818	0.9	0.066666666666667
1	0.1919191919191919	0.9	0.066666666666667
1	0.2020202020202020	0.9	0.066666666666667
1	0.2121212121212121	0.9	0.066666666666667
1	0.2222222222222222	0.9	0.066666666666667
1	0.2323232323232323	0.9	0.066666666666667
1	0.2424242424242424	0.9	0.066666666666667
1	0.2525252525252525	0.9	0.066666666666667
1	0.2626262626262626	0.9	0.066666666666667

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

1	0.272727272727273	0.9	0.066666666666667
1	0.282828282828283	0.9	0.066666666666667
1	0.292929292929293	0.9	0.066666666666667
1	0.303030303030303	0.9	0.066666666666667
1	0.313131313131313	0.9	0.066666666666667
1	0.323232323232323	0.9	0.066666666666667
1	0.333333333333333	0.9	0.066666666666667
1	0.343434343434343	0.9	0.066666666666667
1	0.353535353535354	0.9	0.066666666666667
1	0.363636363636364	0.9	0.066666666666667
1	0.373737373737374	0.9	0.066666666666667
1	0.383838383838384	0.9	0.066666666666667
1	0.393939393939394	0.9	0.066666666666667
1	0.404040404040404	0.9	0.0
1	0.414141414141414	0.9	0.0
1	0.424242424242424	0.9	0.0
1	0.434343434343434	0.9	0.0
1	0.444444444444444	0.9	0.0
1	0.454545454545455	0.9	0.0
1	0.464646464646465	0.9	0.0
1	0.474747474747475	0.9	0.0
1	0.484848484848485	0.9	0.0
1	0.494949494949495	0.9	0.0
1	0.505050505050505	0.9	0.0
1	0.515151515151515	0.9	0.0
1	0.525252525252525	0.9	0.0
1	0.535353535353535	0.9	0.0
1	0.545454545454546	0.9	0.0
1	0.555555555555556	0.9	0.0
1	0.565656565656566	0.9	0.0
1	0.575757575757576	0.9	0.0
1	0.585858585858586	0.9	0.0
1	0.595959595959596	0.9	0.0
1	0.606060606060606	0.9	0.0
1	0.616161616161616	0.9	0.0
1	0.626262626262626	0.9	0.0
1	0.636363636363636	0.9	0.0
1	0.646464646464647	0.9	0.0
1	0.656565656565657	0.9	0.0
1	0.666666666666667	0.9	0.0
1	0.676767676767677	0.9	0.0
1	0.686868686868687	0.9	0.0
1	0.696969696969697	0.9	0.0
1	0.707070707070707	0.9	0.0

# 13: Receiver Operating Characteristic (ROC) (ML Engine)

1	0.7171717171717171	0.9	0.0
1	0.7272727272727272	0.9	0.0
1	0.7373737373737373	0.9	0.0
1	0.7474747474747474	0.9	0.0
1	0.7575757575757575	0.9	0.0
1	0.7676767676767676	0.9	0.0
1	0.7777777777777778	0.9	0.0
1	0.7878787878787878	0.9	0.0
1	0.7979797979797979	0.9	0.0
1	0.8080808080808080	0.9	0.0
1	0.8181818181818181	0.9	0.0
1	0.8282828282828282	0.9	0.0
1	0.8383838383838383	0.9	0.0
1	0.8484848484848484	0.9	0.0
1	0.8585858585858585	0.8	0.0
1	0.8686868686868686	0.8	0.0
1	0.8787878787878787	0.8	0.0
1	0.8888888888888889	0.8	0.0
1	0.8989898989898989	0.8	0.0
1	0.9090909090909090	0.5	0.0
1	0.9191919191919191	0.5	0.0
1	0.9292929292929292	0.5	0.0
1	0.9393939393939393	0.5	0.0
1	0.9494949494949495	0.5	0.0
1	0.9595959595959596	0.5	0.0
1	0.9696969696969697	0.5	0.0
1	0.9797979797979798	0.5	0.0
1	0.9898989898989899	0.5	0.0
1	1.0	0.5	0.0
2	0.0	1.0	1.0
2	0.0101010101010101	1.0	0.4
2	0.0202020202020202	1.0	0.4
2	0.0303030303030303	1.0	0.4
2	0.0404040404040404	1.0	0.4
2	0.0505050505050505	1.0	0.2
2	0.0606060606060606	1.0	0.2
2	0.0707070707070707	1.0	0.2
2	0.0808080808080808	1.0	0.2
2	0.0909090909090909	1.0	0.2
2	0.1010101010101010	1.0	0.1333333333333333
2	0.1111111111111111	1.0	0.1333333333333333
2	0.1212121212121212	1.0	0.1333333333333333
2	0.1313131313131313	1.0	0.1333333333333333
2	0.1414141414141414	1.0	0.1333333333333333

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

2	0.151515151515152	1.0	0.066666666666667
2	0.161616161616162	1.0	0.066666666666667
2	0.171717171717172	1.0	0.066666666666667
2	0.181818181818182	1.0	0.066666666666667
2	0.191919191919192	1.0	0.066666666666667
2	0.202020202020202	1.0	0.066666666666667
2	0.212121212121212	1.0	0.066666666666667
2	0.222222222222222	1.0	0.066666666666667
2	0.232323232323232	1.0	0.066666666666667
2	0.242424242424242	1.0	0.066666666666667
2	0.252525252525253	1.0	0.066666666666667
2	0.262626262626263	1.0	0.066666666666667
2	0.272727272727273	1.0	0.066666666666667
2	0.282828282828283	1.0	0.066666666666667
2	0.292929292929293	1.0	0.066666666666667
2	0.303030303030303	1.0	0.0
2	0.313131313131313	1.0	0.0
2	0.323232323232323	1.0	0.0
2	0.333333333333333	1.0	0.0
2	0.343434343434343	1.0	0.0
2	0.353535353535354	1.0	0.0
2	0.363636363636364	1.0	0.0
2	0.373737373737374	1.0	0.0
2	0.383838383838384	1.0	0.0
2	0.393939393939394	1.0	0.0
2	0.404040404040404	1.0	0.0
2	0.414141414141414	1.0	0.0
2	0.424242424242424	1.0	0.0
2	0.434343434343434	1.0	0.0
2	0.444444444444444	1.0	0.0
2	0.454545454545455	1.0	0.0
2	0.464646464646465	1.0	0.0
2	0.474747474747475	1.0	0.0
2	0.484848484848485	1.0	0.0
2	0.494949494949495	1.0	0.0
2	0.505050505050505	1.0	0.0
2	0.515151515151515	1.0	0.0
2	0.525252525252525	1.0	0.0
2	0.535353535353535	1.0	0.0
2	0.545454545454546	1.0	0.0
2	0.555555555555556	1.0	0.0
2	0.565656565656566	1.0	0.0
2	0.575757575757576	1.0	0.0
2	0.585858585858586	1.0	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

2	0.5959595959595959	1.0	0.0
2	0.6060606060606060	1.0	0.0
2	0.6161616161616161	1.0	0.0
2	0.6262626262626262	1.0	0.0
2	0.6363636363636363	1.0	0.0
2	0.6464646464646464	1.0	0.0
2	0.6565656565656565	1.0	0.0
2	0.6666666666666666	1.0	0.0
2	0.6767676767676767	1.0	0.0
2	0.6868686868686868	1.0	0.0
2	0.6969696969696969	1.0	0.0
2	0.7070707070707070	1.0	0.0
2	0.7171717171717171	1.0	0.0
2	0.7272727272727272	1.0	0.0
2	0.7373737373737373	1.0	0.0
2	0.7474747474747474	1.0	0.0
2	0.7575757575757575	1.0	0.0
2	0.7676767676767676	1.0	0.0
2	0.7777777777777777	1.0	0.0
2	0.7878787878787878	1.0	0.0
2	0.7979797979797979	1.0	0.0
2	0.8080808080808080	1.0	0.0
2	0.8181818181818181	1.0	0.0
2	0.8282828282828282	1.0	0.0
2	0.8383838383838383	1.0	0.0
2	0.8484848484848484	1.0	0.0
2	0.8585858585858585	0.9	0.0
2	0.8686868686868686	0.9	0.0
2	0.8787878787878787	0.9	0.0
2	0.8888888888888888	0.9	0.0
2	0.8989898989898989	0.9	0.0
2	0.9090909090909090	0.7	0.0
2	0.9191919191919191	0.7	0.0
2	0.9292929292929292	0.7	0.0
2	0.9393939393939393	0.7	0.0
2	0.9494949494949495	0.7	0.0
2	0.9595959595959596	0.6	0.0
2	0.9696969696969697	0.6	0.0
2	0.9797979797979798	0.6	0.0
2	0.9898989898989899	0.6	0.0
2	1.0	0.6	0.0
3	0.0	1.0	1.0
3	0.0101010101010101	1.0	0.5833333333333333
3	0.0202020202020202	1.0	0.5833333333333333

# 13: Receiver Operating Characteristic (ROC) (ML Engine)

3 0.0303030303030303	1.0	0.5833333333333333
3 0.0404040404040404	1.0	0.5833333333333333
3 0.0505050505050505	1.0	0.5
3 0.0606060606060606	1.0	0.5
3 0.0707070707070707	1.0	0.5
3 0.0808080808080808	1.0	0.5
3 0.0909090909090909	1.0	0.5
3 0.1010101010101010	1.0	0.4166666666666667
3 0.1111111111111111	1.0	0.4166666666666667
3 0.1212121212121212	1.0	0.4166666666666667
3 0.1313131313131313	1.0	0.4166666666666667
3 0.1414141414141414	1.0	0.4166666666666667
3 0.1515151515151515	1.0	0.25
3 0.1616161616161616	1.0	0.25
3 0.1717171717171717	1.0	0.25
3 0.1818181818181818	1.0	0.25
3 0.1919191919191919	1.0	0.25
3 0.2020202020202020	1.0	0.25
3 0.2121212121212121	1.0	0.25
3 0.2222222222222222	1.0	0.25
3 0.2323232323232323	1.0	0.25
3 0.2424242424242424	1.0	0.25
3 0.2525252525252525	1.0	0.25
3 0.2626262626262626	1.0	0.25
3 0.2727272727272727	1.0	0.25
3 0.2828282828282828	1.0	0.25
3 0.2929292929292929	1.0	0.25
3 0.3030303030303030	1.0	0.25
3 0.3131313131313131	1.0	0.25
3 0.3232323232323232	1.0	0.25
3 0.3333333333333333	1.0	0.25
3 0.3434343434343434	1.0	0.25
3 0.3535353535353535	0.923076923076923	0.1666666666666667
3 0.3636363636363636	0.923076923076923	0.1666666666666667
3 0.3737373737373737	0.923076923076923	0.1666666666666667
3 0.3838383838383838	0.923076923076923	0.1666666666666667
3 0.3939393939393939	0.923076923076923	0.1666666666666667
3 0.4040404040404040	0.923076923076923	0.0833333333333333
3 0.4141414141414141	0.923076923076923	0.0833333333333333
3 0.4242424242424242	0.923076923076923	0.0833333333333333
3 0.4343434343434343	0.923076923076923	0.0833333333333333
3 0.4444444444444444	0.923076923076923	0.0833333333333333
3 0.4545454545454545	0.846153846153846	0.0833333333333333
3 0.4646464646464646	0.846153846153846	0.0833333333333333

3	0.474747474747475	0.846153846153846	0.083333333333333
3	0.484848484848485	0.846153846153846	0.083333333333333
3	0.494949494949495	0.846153846153846	0.083333333333333
3	0.505050505050505	0.846153846153846	0.083333333333333
3	0.515151515151515	0.846153846153846	0.083333333333333
3	0.525252525252525	0.846153846153846	0.083333333333333
3	0.535353535353535	0.846153846153846	0.083333333333333
3	0.545454545454546	0.846153846153846	0.083333333333333
3	0.555555555555556	0.846153846153846	0.083333333333333
3	0.565656565656566	0.846153846153846	0.083333333333333
3	0.575757575757576	0.846153846153846	0.083333333333333
3	0.585858585858586	0.846153846153846	0.083333333333333
3	0.595959595959596	0.846153846153846	0.083333333333333
3	0.606060606060606	0.846153846153846	0.083333333333333
3	0.616161616161616	0.846153846153846	0.083333333333333
3	0.626262626262626	0.846153846153846	0.083333333333333
3	0.636363636363636	0.846153846153846	0.083333333333333
3	0.646464646464647	0.846153846153846	0.083333333333333
3	0.656565656565657	0.846153846153846	0.083333333333333
3	0.666666666666667	0.846153846153846	0.083333333333333
3	0.676767676767677	0.846153846153846	0.083333333333333
3	0.686868686868687	0.846153846153846	0.083333333333333
3	0.696969696969697	0.846153846153846	0.083333333333333
3	0.707070707070707	0.692307692307692	0.083333333333333
3	0.717171717171717	0.692307692307692	0.083333333333333
3	0.727272727272727	0.692307692307692	0.083333333333333
3	0.737373737373737	0.692307692307692	0.083333333333333
3	0.747474747474748	0.692307692307692	0.083333333333333
3	0.757575757575758	0.615384615384615	0.083333333333333
3	0.767676767676768	0.615384615384615	0.083333333333333
3	0.777777777777778	0.615384615384615	0.083333333333333
3	0.787878787878788	0.615384615384615	0.083333333333333
3	0.797979797979798	0.615384615384615	0.083333333333333
3	0.808080808080808	0.615384615384615	0.0
3	0.818181818181818	0.615384615384615	0.0
3	0.828282828282828	0.615384615384615	0.0
3	0.838383838383838	0.615384615384615	0.0
3	0.848484848484849	0.615384615384615	0.0
3	0.858585858585859	0.615384615384615	0.0
3	0.868686868686869	0.615384615384615	0.0
3	0.878787878787879	0.615384615384615	0.0
3	0.888888888888889	0.615384615384615	0.0
3	0.898989898989899	0.615384615384615	0.0
3	0.909090909090909	0.461538461538462	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

[illegible]

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

4	0.353535353535354	1.0	0.0625
4	0.363636363636364	1.0	0.0625
4	0.373737373737374	1.0	0.0625
4	0.383838383838384	1.0	0.0625
4	0.393939393939394	1.0	0.0625
4	0.404040404040404	0.888888888888889	0.0625
4	0.414141414141414	0.888888888888889	0.0625
4	0.424242424242424	0.888888888888889	0.0625
4	0.434343434343434	0.888888888888889	0.0625
4	0.444444444444444	0.888888888888889	0.0625
4	0.454545454545455	0.888888888888889	0.0625
4	0.464646464646465	0.888888888888889	0.0625
4	0.474747474747475	0.888888888888889	0.0625
4	0.484848484848485	0.888888888888889	0.0625
4	0.494949494949495	0.888888888888889	0.0625
4	0.505050505050505	0.888888888888889	0.0625
4	0.515151515151515	0.888888888888889	0.0625
4	0.525252525252525	0.888888888888889	0.0625
4	0.535353535353535	0.888888888888889	0.0625
4	0.545454545454546	0.888888888888889	0.0625
4	0.555555555555556	0.888888888888889	0.0625
4	0.565656565656566	0.888888888888889	0.0625
4	0.575757575757576	0.888888888888889	0.0625
4	0.585858585858586	0.888888888888889	0.0625
4	0.595959595959596	0.888888888888889	0.0625
4	0.606060606060606	0.888888888888889	0.0625
4	0.616161616161616	0.888888888888889	0.0625
4	0.626262626262626	0.888888888888889	0.0625
4	0.636363636363636	0.888888888888889	0.0625
4	0.646464646464647	0.888888888888889	0.0625
4	0.656565656565657	0.888888888888889	0.0625
4	0.666666666666667	0.888888888888889	0.0625
4	0.676767676767677	0.888888888888889	0.0625
4	0.686868686868687	0.888888888888889	0.0625
4	0.696969696969697	0.888888888888889	0.0625
4	0.707070707070707	0.777777777777778	0.0
4	0.717171717171717	0.777777777777778	0.0
4	0.727272727272727	0.777777777777778	0.0
4	0.737373737373737	0.777777777777778	0.0
4	0.747474747474748	0.777777777777778	0.0
4	0.757575757575758	0.777777777777778	0.0
4	0.767676767676768	0.777777777777778	0.0
4	0.777777777777778	0.777777777777778	0.0
4	0.787878787878788	0.777777777777778	0.0

### 13: Receiver Operating Characteristic (ROC) (ML Engine)

4	0.797979797979798	0.777777777777778	0.0
4	0.808080808080808	0.666666666666667	0.0
4	0.818181818181818	0.666666666666667	0.0
4	0.828282828282828	0.666666666666667	0.0
4	0.838383838383838	0.666666666666667	0.0
4	0.848484848484849	0.666666666666667	0.0
4	0.858585858585859	0.555555555555556	0.0
4	0.868686868686869	0.555555555555556	0.0
4	0.878787878787879	0.555555555555556	0.0
4	0.888888888888889	0.555555555555556	0.0
4	0.898989898989899	0.555555555555556	0.0
4	0.909090909090909	0.444444444444444	0.0
4	0.919191919191919	0.444444444444444	0.0
4	0.929292929292929	0.444444444444444	0.0
4	0.939393939393939	0.444444444444444	0.0
4	0.94949494949495	0.444444444444444	0.0
4	0.95959595959596	0.444444444444444	0.0
4	0.96969696969697	0.444444444444444	0.0
4	0.97979797979798	0.444444444444444	0.0
4	0.98989898989899	0.444444444444444	0.0
4	1.0	0.444444444444444	0.0

Download a zip file of all examples and a SQL script file that creates their input tables from the attachment in the left sidebar.

# Receiver operating characteristic

A **receiver operating characteristic curve**, or **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The method was developed for operators of military radar receivers, which is why it is so named.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or *probability of detection*<sup>[7]</sup> in machine learning. The false-positive rate is also known as *probability of false alarm*<sup>[7]</sup> and can be calculated as (1 – specificity). It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity or recall as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from  $-\infty$  to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis.

ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.

The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields and was soon introduced to psychology to account for perceptual detection of stimuli. ROC analysis since then has been used in medicine, radiology, biometrics, forecasting of natural hazards,<sup>[8]</sup> meteorology,<sup>[9]</sup> model performance assessment,<sup>[10]</sup> and other areas for many decades and is increasingly used in machine learning and data mining research.

The ROC is also known as a relative operating characteristic curve, because it is a comparison of two operating characteristics (TPR and FPR) as the criterion changes.<sup>[11]</sup>

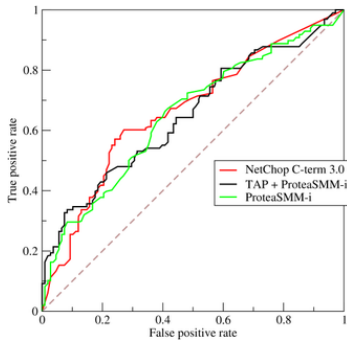
## Contents

- Basic concept**
- ROC space**
- Curves in ROC space**
- Further interpretations**
  - Area under the curve
  - Other measures
- Detection error tradeoff graph**
- Z-score**
- History**
- ROC curves beyond binary classification**
- See also**
- References**
- External links**
- Further reading**

## Basic concept

A classification model (classifier or diagnosis) is a mapping of instances between certain classes/groups. Because the classifier or diagnosis result can be an arbitrary real value (continuous output), the classifier boundary between classes must be determined by a threshold value (for instance, to determine whether a person has hypertension based on a blood pressure measure). Or it can be a discrete class label, indicating one of the classes.

Consider a two-class prediction problem (binary classification), in which the outcomes are labeled either as positive (*p*) or negative (*n*). There are four possible outcomes from a binary classifier. If the outcome from a prediction is *p* and the actual value is also *p*, then it is called a *true positive* (TP); however if the actual value is *n* then it is said to be a *false positive* (FP). Conversely, a *true negative* (TN) has occurred when both the prediction outcome and the actual value are *n*, and *false negative* (FN) is when the prediction outcome is *n* while the actual value is *p*.



ROC curve of three predictors of peptide cleaving in the proteasome.

To get an appropriate example in a real-world problem, consider a diagnostic test that seeks to determine whether a person has a certain disease. A false positive in this case occurs when the person tests positive, but does not actually have the disease. A false negative, on the other hand, occurs when the person tests negative, suggesting they are healthy, when they actually do have the

disease.

Let us define an experiment from **P** positive instances and **N** negative instances for some condition. The four outcomes can be formulated in a  $2 \times 2$  contingency table or confusion matrix, as follows:

Terminology and derivations  
from a confusion matrix

#### condition positive (P)

the number of real positive cases in the data

#### condition negative (N)

the number of real negative cases in the data

#### true positive (TP)

eqv. with hit

#### true negative (TN)

eqv. with correct rejection

#### false positive (FP)

eqv. with false alarm, Type I error

#### false negative (FN)

eqv. with miss, Type II error

#### sensitivity, recall, hit rate, or true positive rate (TPR)

$$\text{TPR} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - \text{FNR}$$

#### specificity, selectivity or true negative rate (TNR)

$$\text{TNR} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - \text{FPR}$$

#### precision or positive predictive value (PPV)

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 1 - \text{FDR}$$

#### negative predictive value (NPV)

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}} = 1 - \text{FOR}$$

#### miss rate or false negative rate (FNR)

$$\text{FNR} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{FN} + \text{TP}} = 1 - \text{TPR}$$

#### fall-out or false positive rate (FPR)

$$\text{FPR} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} = 1 - \text{TNR}$$

#### false discovery rate (FDR)

$$\text{FDR} = \frac{\text{FP}}{\text{FP} + \text{TP}} = 1 - \text{PPV}$$

#### false omission rate (FOR)

$$\text{FOR} = \frac{\text{FN}}{\text{FN} + \text{TN}} = 1 - \text{NPV}$$

#### Prevalence Threshold (PT)

$$\text{PT} = \frac{\sqrt{\text{TPR}(-\text{TNR} + 1)} + \text{TNR} - 1}{(\text{TPR} + \text{TNR} - 1)}$$

#### Threat score (TS) or critical success index (CSI)

$$\text{TS} = \frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}$$

#### accuracy (ACC)

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

#### balanced accuracy (BA)

$$\text{BA} = \frac{\text{TPR} + \text{TNR}}{2}$$

#### F1 score

is the harmonic mean of precision and sensitivity

$$\text{F}_1 = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

#### Matthews correlation coefficient (MCC)

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

**Fowlkes–Mallows index (FM)**

$$FM = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} = \sqrt{PPV \cdot TPR}$$

**informedness or bookmaker informedness (BM)**

$$BM = TPR + TNR - 1$$

**markedness (MK) or deltaP**

$$MK = PPV + NPV - 1$$

Sources: Fawcett (2006),<sup>[1]</sup> Powers (2011),<sup>[2]</sup> Ting (2011),<sup>[3]</sup> and CAWCR<sup>[4]</sup> Chicco & Jurman (2020),<sup>[5]</sup> Tharwat (2018).<sup>[6]</sup>

		True condition			
		Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	<b>True positive</b>	<b>False positive, Type I error</b>	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	<b>False negative, Type II error</b>	<b>True negative</b>	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, $\text{Power} = \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$  $F_1 \text{ score} =$ $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{FNR}{TNR}$	

## ROC space

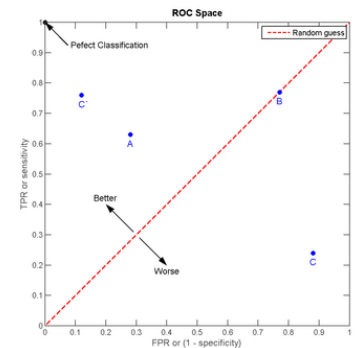
The contingency table can derive several evaluation "metrics" (see infobox). To draw a ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

An ROC space is defined by FPR and TPR as  $x$  and  $y$  axes, respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since TPR is equivalent to sensitivity and FPR is equal to  $1 - \text{specificity}$ , the ROC graph is sometimes called the sensitivity vs  $(1 - \text{specificity})$  plot. Each prediction result or instance of a confusion matrix represents one point in the ROC space.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a *perfect classification*. A random guess would give a point along a diagonal line (the so-called *line of no-discrimination*) from the left bottom to the top right corners (regardless of the positive and negative base rates). An intuitive example of random guessing is a decision by flipping coins. As the size of the sample increases, a random classifier's ROC point tends towards the diagonal line. In the case of a balanced coin, it will tend to the point (0.5, 0.5).

The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random); points below the line represent bad results (worse than random). Note that the output of a consistently bad predictor could simply be inverted to obtain a good predictor.

Let us look into four prediction results from 100 positive and 100 negative instances:



The ROC space and plots of the four prediction examples.

A			B			C			C'		
TP=63	FP=28	91	TP=77	FP=77	154	TP=24	FP=88	112	TP=76	FP=12	88
FN=37	TN=72	109	FN=23	TN=23	46	FN=76	TN=12	88	FN=24	TN=88	112
100	100	200	100	100	200	100	100	200	100	100	200
TPR = 0.63			TPR = 0.77			TPR = 0.24			TPR = 0.76		
FPR = 0.28			FPR = 0.77			FPR = 0.88			FPR = 0.12		
PPV = 0.69			PPV = 0.50			PPV = 0.21			PPV = 0.86		
F1 = 0.66			F1 = 0.61			F1 = 0.23			F1 = 0.81		
ACC = 0.68			ACC = 0.50			ACC = 0.18			ACC = 0.82		

Plots of the four results above in the ROC space are given in the figure. The result of method **A** clearly shows the best predictive power among **A**, **B**, and **C**. The result of **B** lies on the random guess line (the diagonal line), and it can be seen in the table that the accuracy of **B** is 50%. However, when **C** is mirrored across the center point (0.5,0.5), the resulting method **C'** is even better than **A**. This mirrored method simply reverses the predictions of whatever method or test produced the **C** contingency table. Although the original **C** method has negative predictive power, simply reversing its decisions leads to a new predictive method **C'** which has positive predictive power. When the **C** method predicts **p** or **n**, the **C'** method would predict **n** or **p**, respectively. In this manner, the **C'** test would perform the best. The closer a result from a contingency table is to the upper left corner, the better it predicts, but the distance from the random guess line in either direction is the best indicator of how much predictive power a method has. If the result is below the line (i.e. the method is worse than a random guess), all of the method's predictions must be reversed in order to utilize its power, thereby moving the result above the random guess line.

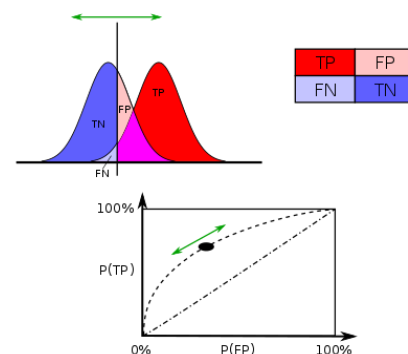
## Curves in ROC space

In binary classification, the class prediction for each instance is often made based on a continuous random variable **X**, which is a "score" computed for the instance (e.g. the estimated probability in logistic regression). Given a threshold parameter **T**, the instance is classified as "positive" if  $X > T$ , and "negative" otherwise. **X** follows a probability density  $f_1(x)$  if the instance actually belongs to class "positive", and  $f_0(x)$

if otherwise. Therefore, the true positive rate is given by  $\text{TPR}(T) = \int_T^\infty f_1(x) dx$

and the false positive rate is given by  $\text{FPR}(T) = \int_T^\infty f_0(x) dx$ . The ROC curve plots parametrically TPR(T) versus FPR(T) with T as the varying parameter.

For example, imagine that the blood protein levels in diseased people and healthy people are normally distributed with means of 2 g/dL and 1 g/dL respectively. A medical test might measure the level of a certain protein in a blood sample and classify any number above a certain threshold as indicating disease. The experimenter can adjust the threshold (black vertical line in the figure), which will in turn change the false positive rate. Increasing the threshold would result in fewer false positives (and more false negatives), corresponding to a leftward movement on the curve. The actual shape of the curve is determined by how much overlap the two distributions have.



## Further interpretations

Sometimes, the ROC is used to generate a summary statistic. Common versions are:

- the intercept of the ROC curve with the line at 45 degrees orthogonal to the no-discrimination line - the balance point where  $\text{Sensitivity} = 1 - \text{Specificity}$
- the intercept of the ROC curve with the tangent at 45 degrees parallel to the no-discrimination line that is closest to the error-free point (0,1) - also called Youden's J statistic and generalized as Informedness
- the area between the ROC curve and the no-discrimination line multiplied by two is called the *Gini coefficient*. It should not be confused with the measure of statistical dispersion also called Gini coefficient.
- the area between the full ROC curve and the triangular ROC curve including only (0,0), (1,1) and one selected operating point (tpr,fpr) - Consistency<sup>[12]</sup>
- the area under the ROC curve, or "AUC" ("Area Under Curve"), or A' (pronounced "a-prime"),<sup>[13]</sup> or "c-statistic" ("concordance statistic").<sup>[14]</sup>

- the sensitivity index  $d'$  (pronounced "d-prime"), the distance between the mean of the distribution of activity in the system under noise-alone conditions and its distribution under signal-alone conditions, divided by their standard deviation, under the assumption that both these distributions are normal with the same standard deviation. Under these assumptions, the shape of the ROC is entirely determined by  $d'$ .

However, any attempt to summarize the ROC curve into a single number loses information about the pattern of tradeoffs of the particular discriminator algorithm.

## Area under the curve

When using normalized units, the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative').<sup>[15]</sup> This can be seen as follows: the area under the curve is given by (the integral boundaries are reversed as large T has a lower value on the x-axis)

$$\begin{aligned} TPR(T) : T &\rightarrow y(x) \\ FPR(T) : T &\rightarrow x \\ A &= \int_{x=0}^1 TPR(FPR^{-1}(x)) dx = \int_{-\infty}^{\infty} TPR(T) FPR'(T) dT = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T) dT' dT = P(X_1 > X_0) \end{aligned}$$

where  $X_1$  is the score for a positive instance and  $X_0$  is the score for a negative instance, and  $f_0$  and  $f_1$  are probability densities as defined in previous section.

It can further be shown that the AUC is closely related to the Mann–Whitney U,<sup>[16][17]</sup> which tests whether positives are ranked higher than negatives. It is also equivalent to the Wilcoxon test of ranks.<sup>[17]</sup> For a predictor  $f$ , an unbiased estimator of its AUC can be expressed by the following *Wilcoxon-Mann-Whitney* statistic<sup>[18]</sup>:

$$AUC(f) = \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \mathbf{1}[f(t_0) < f(t_1)]}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|},$$

where,  $\mathbf{1}[f(t_0) < f(t_1)]$  denotes an *indicator function* which returns 1 iff  $f(t_0) < f(t_1)$  otherwise return 0;  $\mathcal{D}^0$  is the set of negative examples, and  $\mathcal{D}^1$  is the set of positive examples.

The AUC is related to the \*Gini coefficient\* ( $G_1$ ) by the formula  $G_1 = 2AUC - 1$ , where:

$$G_1 = 1 - \sum_{k=1}^n (X_k - X_{k-1})(Y_k + Y_{k-1})^{[19]}$$

In this way, it is possible to calculate the AUC by using an average of a number of trapezoidal approximations.  $G_1$  should not be confused with the measure of statistical dispersion that is also called Gini coefficient.

It is also common to calculate the Area Under the ROC Convex Hull (ROC AUCH = ROCH AUC) as any point on the line segment between two prediction results can be achieved by randomly using one or the other system with probabilities proportional to the relative length of the opposite component of the segment.<sup>[20]</sup> It is also possible to invert concavities – just as in the figure the worse solution can be reflected to become a better solution; concavities can be reflected in any line segment, but this more extreme form of fusion is much more likely to overfit the data.<sup>[21]</sup>

The machine learning community most often uses the ROC AUC statistic for model comparison.<sup>[22]</sup> This practice has been questioned because AUC estimates are quite noisy and suffer from other problems.<sup>[23][24][25]</sup> Nonetheless, the coherence of AUC as a measure of aggregated classification performance has been vindicated, in terms of a uniform rate distribution,<sup>[26]</sup> and AUC has been linked to a number of other performance metrics such as the Brier score.<sup>[27]</sup>

Another problem with ROC AUC is that reducing the ROC Curve to a single number ignores the fact that it is about the tradeoffs between the different systems or performance points plotted and not the performance of an individual system, as well as ignoring the possibility of concavity repair, so that related alternative measures such as Informedness or DeltaP are recommended.<sup>[12][28]</sup> These measures are essentially equivalent to the Gini for a single prediction point with DeltaP' = Informedness = 2AUC-1, whilst DeltaP = Markedness represents the dual (viz. predicting the prediction from the real class) and their geometric mean is the Matthews correlation coefficient.

Whereas ROC AUC varies between 0 and 1 – with an uninformative classifier yielding 0.5 – the alternative measures known as Informedness, Certainty <sup>[12]</sup> and Gini Coefficient (in the single parameterization or single system case) all have the advantage that 0 represents chance performance whilst 1 represents perfect performance, and -1 represents the "perverse" case of full informedness

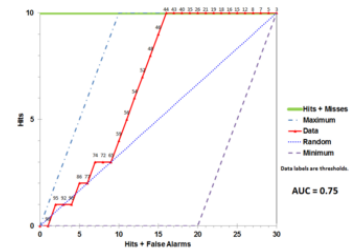
Sometimes it can be more useful to look at a specific region of the ROC Curve rather than at the whole curve. It is possible to compute partial AUC.<sup>[31]</sup> For example, one could focus on the region of the curve with low false positive rate, which is often of prime interest for population screening tests.<sup>[32]</sup> Another common approach for classification problems in which  $P \ll N$  (common in bioinformatics applications) is to use a logarithmic scale for the x-axis.<sup>[33]</sup>

The ROC area under the curve is also called **c-statistic** or **c statistic**.<sup>[34]</sup>

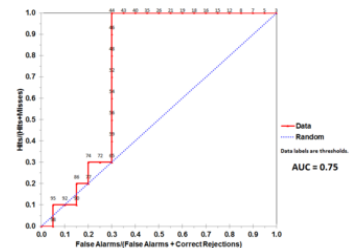
Other measures

The Total Operating Characteristic (TOC) also characterizes diagnostic ability while revealing more information than the ROC. For each threshold, ROC reveals two ratios,  $TP/(TP + FN)$  and  $FP/(FP + TN)$ . In other words, ROC reveals hits/(hits + misses) and false alarms/(false alarms + correct rejections). On the other hand, TOC shows the total information in the contingency table for each threshold.<sup>[35]</sup> The TOC method reveals all of the information that the ROC method provides, plus additional important information that ROC does not reveal, i.e. the size of every entry in the contingency table for each threshold. TOC also provides the popular AUC of the ROC.<sup>[36]</sup>

These figures are the TOC and ROC curves using the same data and thresholds. Consider the point that corresponds to a threshold of 74. The TOC curve shows the number of hits, which is 3, and hence the number of misses, which is 7. Additionally, the TOC curve shows that the number of false alarms is 4 and the number of correct rejections is 16. At any given point in the ROC curve, it is possible to glean values for the ratios of false alarms/(false alarms + correct rejections) and hits/(hits + misses). For example, at threshold 74, it is evident that the x coordinate is 0.2 and the y coordinate is 0.3. However, these two values are insufficient to construct all entries of the underlying two-by-two contingency table.



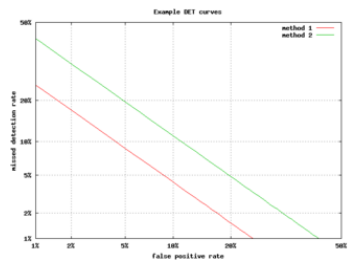
TOC Curve



ROC Curve

Detection error tradeoff graph

An alternative to the ROC curve is the detection error tradeoff (DET) graph, which plots the false negative rate (missed detections) vs. the false positive rate (false alarms) on non-linearly transformed x- and y-axes. The transformation function is the quantile function of the normal distribution, i.e., the inverse of the cumulative normal distribution. It is, in fact, the same transformation as zROC, below, except that the complement of the hit rate, the miss rate or false negative rate, is used. This alternative spends more graph area on the region of interest. Most of the ROC area is of little interest; one primarily cares about the region tight against the y-axis and the top left corner – which, because of using miss rate instead of its complement, the hit rate, is the lower left corner in a DET plot. Furthermore, DET graphs have the useful property of linearity and a linear threshold behavior for normal distributions.<sup>[37]</sup> The DET plot is used extensively in the automatic speaker recognition community, where the name DET was first used. The analysis of the ROC performance in graphs with this warping of the axes was used by psychologists in perception studies halfway through the 20th century, where this was dubbed "double probability paper".<sup>[38]</sup>



Example DET graph

Z-score

If a standard score is applied to the ROC curve, the curve will be transformed into a straight line.<sup>[39]</sup> This z-score is based on a normal distribution with a mean of zero and a standard deviation of one. In memory strength theory, one must assume that the zROC is not only linear, but has a slope of 1.0. The normal distributions of targets (studied objects that the subjects need to recall) and lures (non studied objects that the subjects attempt to recall) is the factor causing the zROC to be linear.

The linearity of the zROC curve depends on the standard deviations of the target and lure strength distributions. If the standard deviations are equal, the slope will be 1.0. If the standard deviation of the target strength distribution is larger than the standard deviation of the lure strength distribution, then the slope will be smaller than 1.0. In most studies, it has been found that the zROC

curve slopes constantly fall below 1, usually between 0.5 and 0.9.<sup>[40]</sup> Many experiments yielded a zROC slope of 0.8. A slope of 0.8 implies that the variability of the target strength distribution is 25% larger than the variability of the lure strength distribution.<sup>[41]</sup>

Another variable used is  $d'$  (d prime) (discussed above in "Other measures"), which can easily be expressed in terms of z-values. Although  $d'$  is a commonly used parameter, it must be recognized that it is only relevant when strictly adhering to the very strong assumptions of strength theory made above.<sup>[42]</sup>

The z-score of an ROC curve is always linear, as assumed, except in special situations. The Yonelinas familiarity-recollection model is a two-dimensional account of recognition memory. Instead of the subject simply answering yes or no to a specific input, the subject gives the input a feeling of familiarity, which operates like the original ROC curve. What changes, though, is a parameter for Recollection (R). Recollection is assumed to be all-or-none, and it trumps familiarity. If there were no recollection component, zROC would have a predicted slope of 1. However, when adding the recollection component, the zROC curve will be concave up, with a decreased slope. This difference in shape and slope result from an added element of variability due to some items being recollected. Patients with anterograde amnesia are unable to recollect, so their Yonelinas zROC curve would have a slope close to 1.0.<sup>[43]</sup>

## History

The ROC curve was first used during World War II for the analysis of radar signals before it was employed in signal detection theory.<sup>[44]</sup> Following the attack on Pearl Harbor in 1941, the United States army began new research to increase the prediction of correctly detected Japanese aircraft from their radar signals. For these purposes they measured the ability of a radar receiver operator to make these important distinctions, which was called the Receiver Operating Characteristic.<sup>[45]</sup>

In the 1950s, ROC curves were employed in psychophysics to assess human (and occasionally non-human animal) detection of weak signals.<sup>[44]</sup> In medicine, ROC analysis has been extensively used in the evaluation of diagnostic tests.<sup>[46][47]</sup> ROC curves are also used extensively in epidemiology and medical research and are frequently mentioned in conjunction with evidence-based medicine. In radiology, ROC analysis is a common technique to evaluate new radiology techniques.<sup>[48]</sup> In the social sciences, ROC analysis is often called the ROC Accuracy Ratio, a common technique for judging the accuracy of default probability models. ROC curves are widely used in laboratory medicine to assess the diagnostic accuracy of a test, to choose the optimal cut-off of a test and to compare diagnostic accuracy of several tests.

ROC curves also proved useful for the evaluation of machine learning techniques. The first application of ROC in machine learning was by Spackman who demonstrated the value of ROC curves in comparing and evaluating different classification algorithms.<sup>[49]</sup>

ROC curves are also used in verification of forecasts in meteorology.<sup>[50]</sup>

## ROC curves beyond binary classification

The extension of ROC curves for classification problems with more than two classes has always been cumbersome, as the degrees of freedom increase quadratically with the number of classes, and the ROC space has  $c(c - 1)$  dimensions, where  $c$  is the number of classes.<sup>[51]</sup> Some approaches have been made for the particular case with three classes (three-way ROC).<sup>[52]</sup> The calculation of the volume under the ROC surface (VUS) has been analyzed and studied as a performance metric for multi-class problems.<sup>[53]</sup> However, because of the complexity of approximating the true VUS, some other approaches <sup>[54]</sup> based on an extension of AUC are more popular as an evaluation metric.

Given the success of ROC curves for the assessment of classification models, the extension of ROC curves for other supervised tasks has also been investigated. Notable proposals for regression problems are the so-called regression error characteristic (REC) Curves <sup>[55]</sup> and the Regression ROC (RROC) curves.<sup>[56]</sup> In the latter, RROC curves become extremely similar to ROC curves for classification, with the notions of asymmetry, dominance and convex hull. Also, the area under RROC curves is proportional to the error variance of the regression model.

## See also

- Brier score
- Coefficient of determination
- Constant false alarm rate
- Detection error tradeoff
- Detection theory
- F1 score
- False alarm
- Precision and recall
- ROCET
- Total operating characteristic

## References

1. Fawcett, Tom (2006). "An Introduction to ROC Analysis" (<http://people.inf.elte.hu/kiss/11dwhdm/roc.pdf>) (PDF). *Pattern Recognition Letters*. **27** (8): 861–874. doi:10.1016/j.patrec.2005.10.010 (<https://doi.org/10.1016%2Fj.patrec.2005.10.010>).
2. Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation" (<https://www.researchgate.net/publication/228529307>). *Journal of Machine Learning Technologies*. **2** (1): 37–63.
3. Ting, Kai Ming (2011). Sammut, Claude; Webb, Geoffrey I (eds.). *Encyclopedia of machine learning*. Springer. doi:10.1007/978-0-387-30164-8 (<https://doi.org/10.1007%2F978-0-387-30164-8>). ISBN 978-0-387-30164-8.
4. Brooks, Harold; Brown, Barb; Ebert, Beth; Ferro, Chris; Jolliffe, Ian; Koh, Tieh-Yong; Roebber, Paul; Stephenson, David (2015-01-26). "WWRP/WGNE Joint Working Group on Forecast Verification Research" (<https://www.cawcr.gov.au/projects/verification/>). *Collaboration for Australian Weather and Climate Research*. World Meteorological Organisation. Retrieved 2019-07-17.
5. Chicco D, Jurman G (January 2020). "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation" (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6941312>). *BMC Genomics*. **21** (1): 6-1–6-13. doi:10.1186/s12864-019-6413-7 (<https://doi.org/10.1186%2Fs12864-019-6413-7>). PMC 6941312 (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6941312>). PMID 31898477 (<https://pubmed.ncbi.nlm.nih.gov/31898477>).
6. Tharwat A (August 2018). "Classification assessment methods" (<https://doi.org/10.1016/j.aci.2018.08.003>). *Applied Computing and Informatics*. doi:10.1016/j.aci.2018.08.003 (<https://doi.org/10.1016%2Fj.aci.2018.08.003>).
7. "Detector Performance Analysis Using ROC Curves - MATLAB & Simulink Example" (<http://www.mathworks.com/help/phased/examples/detector-performance-analysis-using-roc-curves.html>). *www.mathworks.com*. Retrieved 11 August 2016.
8. Peres, D. J.; Cancelliere, A. (2014-12-08). "Derivation and evaluation of landslide-triggering thresholds by a Monte Carlo approach" (<https://doi.org/10.5194/hess-18-4913-2014>). *Hydrol. Earth Syst. Sci.* **18** (12): 4913–4931. Bibcode:2014HESS...18.4913P (<https://ui.adsabs.harvard.edu/abs/2014HESS...18.4913P>). doi:10.5194/hess-18-4913-2014 (<https://doi.org/10.5194%2Fhess-18-4913-2014>). ISSN 1607-7938 (<https://www.worldcat.org/issn/1607-7938>).
9. Murphy, Allan H. (1996-03-01). <0003:tfaase>2.0.co;2 "The Finley Affair: A Signal Event in the History of Forecast Verification" ([https://doi.org/10.1175/1520-0434\(1996\)011](https://doi.org/10.1175/1520-0434(1996)011)). *Weather and Forecasting*. **11** (1): 3–20. Bibcode:1996WtFor..11....3M (<https://ui.adsabs.harvard.edu/abs/1996WtFor..11....3M>). doi:10.1175/1520-0434(1996)011<0003:tfaase>2.0.co;2 (<https://doi.org/10.1175%2F1520-0434%281996%29011%3C0003%3Atfaase%3E2.0.co%3B2>). ISSN 0882-8156 (<https://www.worldcat.org/issn/0882-8156>).
10. Peres, D. J.; Iuppa, C.; Cavallaro, L.; Cancelliere, A.; Foti, E. (2015-10-01). "Significant wave height record extension by neural networks and reanalysis wind data". *Ocean Modelling*. **94**: 128–140. Bibcode:2015OcMod..94..128P (<https://ui.adsabs.harvard.edu/abs/2015OcMod..94..128P>). doi:10.1016/j.ocemod.2015.08.002 (<https://doi.org/10.1016%2Fj.ocemod.2015.08.002>).
11. Swets, John A.; *Signal detection theory and ROC analysis in psychology and diagnostics : collected papers* (<https://www.questia.com/PM.qst?a=o&d=91082370>). Lawrence Erlbaum Associates, Mahwah, NJ, 1996
12. Powers, David MW (2012). "ROC-ConCert: ROC-Based Measurement of Consistency and Certainty" (<http://www.academia.edu/download/31939951/201203-SCET30795-ROC-ConCert-PID1124774.pdf>) (PDF). *Spring Congress on Engineering and Technology (SCET)*. **2**. IEEE. pp. 238–241.
13. Fogarty, James; Baker, Ryan S.; Hudson, Scott E. (2005). "Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction" (<http://portal.acm.org/citation.cfm?id=1089530>). *ACM International Conference Proceeding Series, Proceedings of Graphics Interface 2005*. Waterloo, ON: Canadian Human-Computer Communications Society.
14. Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (2nd ed.).
15. Fawcett, Tom (2006); *An introduction to ROC analysis* (<https://www.math.ucdavis.edu/~saito/data/roc/fawcett-roc.pdf>), *Pattern Recognition Letters*, 27, 861–874.
16. Hanley, James A.; McNeil, Barbara J. (1982). "The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve". *Radiology*. **143** (1): 29–36. doi:10.1148/radiology.143.1.7063747 (<https://doi.org/10.1148%2Fradiology.143.1.7063747>). PMID 7063747 (<https://pubmed.ncbi.nlm.nih.gov/7063747>). S2CID 10511727 (<https://api.semanticscholar.org/CorpusID:10511727>).
17. Mason, Simon J.; Graham, Nicholas E. (2002). "Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation" ([https://web.archive.org/web/20081120134338/http://www.inmet.gov.br/documentos/cursol\\_INMET\\_IRI/Climate\\_Information\\_Course/References/Mason%2BGraham\\_2002.pdf](https://web.archive.org/web/20081120134338/http://www.inmet.gov.br/documentos/cursol_INMET_IRI/Climate_Information_Course/References/Mason%2BGraham_2002.pdf)) (PDF). *Quarterly Journal of the Royal Meteorological Society*. **128** (584): 2145–2166. Bibcode:2002QJRMS.128.2145M (<https://ui.adsabs.harvard.edu/abs/2002QJRMS.128.2145M>). CiteSeerX 10.1.1.458.8392 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.458.8392>). doi:10.1256/003590002320603584 (<https://doi.org/10.1256%2F003590002320603584>). Archived from the original ([http://www.inmet.gov.br/documentos/cursol\\_INMET\\_IRI/Climate\\_Information\\_Course/References/Mason+Graham\\_2002.pdf](http://www.inmet.gov.br/documentos/cursol_INMET_IRI/Climate_Information_Course/References/Mason+Graham_2002.pdf)) (PDF) on 2008-11-20.
18. Calders, Toon; Jaroszewicz, Szymon (2007). Kok, Joost N.; Koronacki, Jacek; Lopez de Mantaras, Ramon; Matwin, Stan; Mladenić, Dunja; Skowron, Andrzej (eds.). "Efficient AUC Optimization for Classification" ([https://doi.org/10.1007/978-3-540-7497-6\\_9](https://doi.org/10.1007/978-3-540-7497-6_9)). *Knowledge Discovery in Databases: PKDD 2007. Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer. **4702**: 42–53. doi:10.1007/978-3-540-74976-9\_8 ([https://doi.org/10.1007%2F978-3-540-74976-9\\_8](https://doi.org/10.1007%2F978-3-540-74976-9_8)). ISBN 978-3-540-74976-9.
19. Hand, David J.; and Till, Robert J. (2001); *A simple generalization of the area under the ROC curve for multiple class classification problems*, *Machine Learning*, 45, 171–186.
20. Provost, F.; Fawcett, T. (2001). "Robust classification for imprecise environments". *Machine Learning*. **42** (3): 203–231. arXiv:cs/0009007 (<https://arxiv.org/abs/cs/0009007>). doi:10.1023/a:1007601015854 (<https://doi.org/10.1023%2Fa%3A1007601015854>).

21. Flach, P.A.; Wu, S. (2005). "Repairing concavities in ROC curves." ([http://www.icml-2011.org/papers/385\\_icmlpaper.pdf](http://www.icml-2011.org/papers/385_icmlpaper.pdf)) (PDF). *19th International Joint Conference on Artificial Intelligence (IJCAI/05)*. pp. 702–707.
22. Hanley, James A.; McNeil, Barbara J. (1983-09-01). "A method of comparing the areas under receiver operating characteristic curves derived from the same cases" (<https://doi.org/10.1148/radiology.148.3.6878708>). *Radiology*. **148** (3): 839–843. doi:10.1148/radiology.148.3.6878708 (<https://doi.org/10.1148%2Fradiology.148.3.6878708>). PMID 6878708 (<https://pubmed.ncbi.nlm.nih.gov/6878708/>).
23. Hanczar, Blaise; Hua, Jianping; Sima, Chao; Weinstein, John; Bittner, Michael; Dougherty, Edward R (2010). "Small-sample precision of ROC-related estimates" (<https://doi.org/10.1093/bioinformatics/btq037>). *Bioinformatics*. **26** (6): 822–830. doi:10.1093/bioinformatics/btq037 (<https://doi.org/10.1093%2Fbioinformatics%2Fbtq037>). PMID 20130029 (<https://pubmed.ncbi.nlm.nih.gov/20130029/>).
24. Lobo, Jorge M.; Jiménez-Valverde, Alberto; Real, Raimundo (2008). "AUC: a misleading measure of the performance of predictive distribution models". *Global Ecology and Biogeography*. **17** (2): 145–151. doi:10.1111/j.1466-8238.2007.00358.x (<http://doi.org/10.1111%2Fj.1466-8238.2007.00358.x>). S2CID 15206363 (<https://api.semanticscholar.org/CorpusID:15206363>).
25. Hand, David J (2009). "Measuring classifier performance: A coherent alternative to the area under the ROC curve" (<https://doi.org/10.1007/s10994-009-5119-5>). *Machine Learning*. **77**: 103–123. doi:10.1007/s10994-009-5119-5 (<https://doi.org/10.1007%2Fs10994-009-5119-5>).
26. Flach, P.A.; Hernandez-Orallo, J.; Ferri, C. (2011). "A coherent interpretation of AUC as a measure of aggregated classification performance." ([http://www.icml-2011.org/papers/385\\_icmlpaper.pdf](http://www.icml-2011.org/papers/385_icmlpaper.pdf)) (PDF). *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pp. 657–664.
27. Hernandez-Orallo, J.; Flach, P.A.; Ferri, C. (2012). "A unified view of performance metrics: translating threshold choice into expected classification loss" (<http://jmlr.org/papers/volume13/hernandez-orallo12a/hernandez-orallo12a.pdf>) (PDF). *Journal of Machine Learning Research*. **13**: 2813–2869.
28. Powers, David M.W. (2012). "The Problem of Area Under the Curve". *International Conference on Information Science and Technology*.
29. Powers, David M. W. (2003). "Recall and Precision versus the Bookmaker" (<https://dl.dropbox.com/u/27743223/200302-ICCS-Bookmaker.pdf>) (PDF). *Proceedings of the International Conference on Cognitive Science (ICCS-2003)*, Sydney Australia, 2003, pp. 529–534.
30. Powers, David M. W. (2012). "The Problem with Kappa" (<http://arquivo.pt/wayback/20160518183306/http://dl.dropbox.com/u/27743223/201209-eacl2012-Kappa.pdf>) (PDF). *Conference of the European Chapter of the Association for Computational Linguistics (EACL2012) Joint ROBUST-UNSUP Workshop*. Archived from the original (<https://dl.dropbox.com/u/27743223/201209-eacl2012-Kappa.pdf>) (PDF) on 2016-05-18. Retrieved 2012-07-20.
31. McClish, Donna Katzman (1989-08-01). "Analyzing a Portion of the ROC Curve". *Medical Decision Making*. **9** (3): 190–195. doi:10.1177/0272989X8900900307 (<https://doi.org/10.1177%2F0272989X8900900307>). PMID 2668680 (<https://pubmed.ncbi.nlm.nih.gov/2668680/>). S2CID 24442201 (<https://api.semanticscholar.org/CorpusID:24442201>).
32. Dodd, Lori E.; Pepe, Margaret S. (2003). "Partial AUC Estimation and Regression" (<http://biostats.bepress.com/cgi/viewcontent.cgi?article=1005&context=uwbiostat>). *Biometrics*. **59** (3): 614–623. doi:10.1111/1541-0420.00071 (<https://doi.org/10.1111%2F1541-0420.00071>). PMID 14601762 (<https://pubmed.ncbi.nlm.nih.gov/14601762/>).
33. Karplus, Kevin (2011); *Better than Chance: the importance of null models* (<http://www.soe.ucsc.edu/~karplus/papers/better-than-chance-sep-07.pdf>), University of California, Santa Cruz, in Proceedings of the First International Workshop on Pattern Recognition in Proteomics, Structural Biology and Bioinformatics (PR PS BB 2011)
34. "C-Statistic: Definition, Examples, Weighting and Significance" (<https://www.statisticshowto.datasciencecentral.com/c-statistic/>). *Statistics How To*. August 28, 2016.
35. Pontius, Robert Gilmore; Parmentier, Benoit (2014). "Recommendations for using the Relative Operating Characteristic (ROC)". *Landscape Ecology*. **29** (3): 367–382. doi:10.1007/s10980-013-9984-8 (<https://doi.org/10.1007%2Fs10980-013-9984-8>). S2CID 15924380 (<https://api.semanticscholar.org/CorpusID:15924380>).
36. Pontius, Robert Gilmore; Si, Kangping (2014). "The total operating characteristic to measure diagnostic ability for multiple thresholds". *International Journal of Geographical Information Science*. **28** (3): 570–583. doi:10.1080/13658816.2013.862623 (<https://doi.org/10.1080%2F13658816.2013.862623>). S2CID 29204880 (<https://api.semanticscholar.org/CorpusID:29204880>).
37. Navratil, J.; Klusacek, D. (2007-04-01). *On Linear DETs*. 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07. 4. pp. IV–229–IV–232. doi:10.1109/ICASSP.2007.367205 (<https://doi.org/10.1109%2FICASSP.2007.367205>). ISBN 978-1-4244-0727-9. S2CID 18173315 (<https://api.semanticscholar.org/CorpusID:18173315>).
38. Dev P. Chakraborty (December 14, 2017). "double+probability+paper" *Observer Performance Methods for Diagnostic Imaging: Foundations, Modeling, and Applications with R-Based Examples* (<https://books.google.com/books?id=MwZDDwAAQBAJ&pg=PT214&lpg=PT214&dq=>). CRC Press. p. 214. ISBN 9781351230711. Retrieved July 11, 2019.
39. MacMillan, Neil A.; Creelman, C. Douglas (2005). *Detection Theory: A User's Guide* (2nd ed.). Mahwah, NJ: Lawrence Erlbaum Associates. ISBN 978-1-4106-1114-7.
40. Glanzer, Murray; Kisok, Kim; Hilford, Andy; Adams, John K. (1999). "Slope of the receiver-operating characteristic in recognition memory". *Journal of Experimental Psychology: Learning, Memory, and Cognition*. **25** (2): 500–513. doi:10.1037/0278-7393.25.2.500 (<https://doi.org/10.1037%2F0278-7393.25.2.500>).
41. Ratcliff, Roger; McCoon, Gail; Tindall, Michael (1994). "Empirical generality of data from recognition memory ROC functions and implications for GMMs". *Journal of Experimental Psychology: Learning, Memory, and Cognition*. **20** (4): 763–785. CiteSeerX 10.1.1.410.2114 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.410.2114>). doi:10.1037/0278-7393.20.4.763 (<https://doi.org/10.1037%2F0278-7393.20.4.763>).

42. Zhang, Jun; Mueller, Shane T. (2005). "A note on ROC analysis and non-parametric estimate of sensitivity". *Psychometrika*. **70**: 203–212. CiteSeerX 10.1.1.162.1515 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.162.1515>). doi:10.1007/s11336-003-1119-8 (<https://doi.org/10.1007%2Fs11336-003-1119-8>). S2CID 122355230 (<https://api.semanticscholar.org/CorpusID:122355230>).
43. Yonelinas, Andrew P.; Kroll, Neal E. A.; Dobbins, Ian G.; Lazzara, Michele; Knight, Robert T. (1998). "Recollection and familiarity deficits in amnesia: Convergence of remember-know, process dissociation, and receiver operating characteristic data". *Neuropsychology*. **12** (3): 323–339. doi:10.1037/0894-4105.12.3.323 (<https://doi.org/10.1037%2F0894-4105.12.3.323>). PMID 9673991 (<https://pubmed.ncbi.nlm.nih.gov/9673991/>).
44. Green, David M.; Swets, John A. (1966). *Signal detection theory and psychophysics*. New York, NY: John Wiley and Sons Inc. ISBN 978-0-471-32420-1.
45. "Using the Receiver Operating Characteristic (ROC) curve to analyze a classification model: A final note of historical interest" (<http://www.math.utah.edu/~gamez/files/ROC-Curves.pdf>) (PDF). *Department of Mathematics, University of Utah*. Department of Mathematics, University of Utah. Retrieved May 25, 2017.
46. Zweig, Mark H.; Campbell, Gregory (1993). "Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine" (<http://www.clinchem.org/content/39/4/561.full.pdf>) (PDF). *Clinical Chemistry*. **39** (8): 561–577. doi:10.1093/clinchem/39.4.561 (<https://doi.org/10.1093%2Fclinchem%2F39.4.561>). PMID 8472349 (<https://pubmed.ncbi.nlm.nih.gov/8472349/>).
47. Pepe, Margaret S. (2003). *The statistical evaluation of medical tests for classification and prediction*. New York, NY: Oxford. ISBN 978-0-19-856582-6.
48. Obuchowski, Nancy A. (2003). "Receiver operating characteristic curves and their use in radiology". *Radiology*. **229** (1): 3–8. doi:10.1148/radiol.2291010898 (<https://doi.org/10.1148%2Fradol.2291010898>). PMID 14519861 (<https://pubmed.ncbi.nlm.nih.gov/14519861/>).
49. Spackman, Kent A. (1989). "Signal detection theory: Valuable tools for evaluating inductive learning". *Proceedings of the Sixth International Workshop on Machine Learning*. San Mateo, CA: Morgan Kaufmann. pp. 160–163.
50. Kharin, Viatcheslav (2003). <4145:OTRSOP>2.0.CO;2 "On the ROC score of probability forecasts" ([https://doi.org/10.1175/1520-0442\(2003\)016](https://doi.org/10.1175/1520-0442(2003)016)). *Journal of Climate*. **16** (24): 4145–4150. Bibcode:2003JCLI...16.4145K (<https://ui.adsabs.harvard.edu/abs/2003JCLI...16.4145K>). doi:10.1175/1520-0442(2003)016<4145:OTRSOP>2.0.CO;2 (<https://doi.org/10.1175%2F1520-0442%282003%29016%3C4145%3AOTRSOP%3E2.0.CO%3B2>).
51. Srinivasan, A. (1999). "Note on the Location of Optimal Classifiers in N-dimensional ROC Space". *Technical Report PRG-TR-2-99, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford*. CiteSeerX 10.1.1.35.703 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.703>).
52. Mossman, D. (1999). "Three-way ROCs". *Medical Decision Making*. **19** (1): 78–89. doi:10.1177/0272989x9901900110 (<https://doi.org/10.1177%2F0272989x9901900110>). PMID 9917023 (<https://pubmed.ncbi.nlm.nih.gov/9917023/>).
53. Ferri, C.; Hernandez-Orallo, J.; Salido, M.A. (2003). "Volume under the ROC Surface for Multi-class Problems". *Machine Learning: ECML 2003*. pp. 108–120.
54. Till, D.J.; Hand, R.J. (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems" (<https://doi.org/10.1023/A:1010920819831>). *Machine Learning*. **45** (2): 171–186. doi:10.1023/A:1010920819831 (<https://doi.org/10.1023%2FA%3A1010920819831>).
55. Bi, J.; Bennett, K.P. (2003). "Regression error characteristic curves" (<https://www.aiai.org/Papers/ICML/2003/ICML03-009.pdf>) (PDF). *Twentieth International Conference on Machine Learning (ICML-2003)*. Washington, DC.
56. Hernandez-Orallo, J. (2013). "ROC curves for regression". *Pattern Recognition*. **46** (12): 3395–3411. doi:10.1016/j.patcog.2013.06.014 (<https://doi.org/10.1016%2Fj.patcog.2013.06.014>). hdl:10251/40252 (<https://hdl.handle.net/10251%2F40252>).

## External links

- ROC demo (<https://kennis-research.shinyapps.io/ROC-Curves/>)
- another ROC demo (<http://www.navan.name/roc/>)
- ROC video explanation (<https://www.youtube.com/watch?v=OA16eAyP-yo>)
- An Introduction to the Total Operating Characteristic: Utility in Land Change Model Evaluation (<https://www.youtube.com/watch?v=KKVC3GT5EPw>)
- How to run the TOC Package in R (<https://www.youtube.com/watch?v=1JRwVOi0FSE>)
- TOC R package on Github (<https://github.com/amsantac/TOC>)
- Excel Workbook for generating TOC curves (<http://www2.clarku.edu/~rpontius/TOCexample2.xlsx>)

## Further reading

- Balakrishnan, Narayanaswamy (1991); *Handbook of the Logistic Distribution*, Marcel Dekker, Inc., ISBN 978-0-8247-8587-1
- Brown, Christopher D.; Davis, Herbert T. (2006). "Receiver operating characteristic curves and related decision measures: a tutorial". *Chemometrics and Intelligent Laboratory Systems*. **80**: 24–38. doi:10.1016/j.chemolab.2005.05.004 (<https://doi.org/10.1016%2Fj.chemolab.2005.05.004>).

- Rotello, Caren M.; Heit, Evan; Dubé, Chad (2014). "When more data steer us wrong: replications with the wrong dependent measure perpetuate erroneous conclusions" ([http://faculty.ucmerced.edu/sites/default/files/eheit/files/rotello\\_heit\\_dube\\_pbr.pdf](http://faculty.ucmerced.edu/sites/default/files/eheit/files/rotello_heit_dube_pbr.pdf)) (PDF). *Psychonomic Bulletin & Review*. **22** (4): 944–954. doi:10.3758/s13423-014-0759-2 (<https://doi.org/10.3758/s13423-014-0759-2>). PMID 25384892 (<https://pubmed.ncbi.nlm.nih.gov/25384892>). S2CID 6046065 (<https://api.semanticscholar.org/CorpusID:6046065>).
- Fawcett, Tom (2004). "ROC Graphs: Notes and Practical Considerations for Researchers" ([http://home.comcast.net/~tom.fawcett/public\\_html/papers/ROC101.pdf](http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf)) (PDF). *Pattern Recognition Letters*. **27** (8): 882–891. CiteSeerX 10.1.1.145.4649 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.4649>). doi:10.1016/j.patrec.2005.10.012 (<https://doi.org/10.1016/j.patrec.2005.10.012>).
- Gonen, Mithat (2007); *Analyzing Receiver Operating Characteristic Curves Using SAS*, SAS Press, ISBN 978-1-59994-298-8
- Green, William H., (2003) *Econometric Analysis*, fifth edition, Prentice Hall, ISBN 0-13-066189-9
- Heagerty, Patrick J.; Lumley, Thomas; Pepe, Margaret S. (2000). "Time-dependent ROC Curves for Censored Survival Data and a Diagnostic Marker". *Biometrics*. **56** (2): 337–344. doi:10.1111/j.0006-341x.2000.00337.x (<https://doi.org/10.1111/j.0006-341x.2000.00337.x>). PMID 10877287 (<https://pubmed.ncbi.nlm.nih.gov/10877287>). S2CID 8822160 (<https://api.semanticscholar.org/CorpusID:8822160>).
- Hosmer, David W.; and Lemeshow, Stanley (2000); *Applied Logistic Regression*, 2nd ed., New York, NY: Wiley, ISBN 0-471-35632-8
- Lasko, Thomas A.; Bhagwat, Jui G.; Zou, Kelly H.; Ohno-Machado, Lucila (2005). "The use of receiver operating characteristic curves in biomedical informatics". *Journal of Biomedical Informatics*. **38** (5): 404–415. CiteSeerX 10.1.1.97.9674 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.9674>). doi:10.1016/j.jbi.2005.02.008 (<https://doi.org/10.1016/j.jbi.2005.02.008>). PMID 16198999 (<https://pubmed.ncbi.nlm.nih.gov/16198999>).
- Mas, Jean-François; Filho, Britaldo Soares; Pontius, Jr, Robert Gilmore; Gutiérrez, Michelle Farfán; Rodrigues, Hermann (2013). "A suite of tools for ROC analysis of spatial models" (<https://doi.org/10.3390/ijgi2030869>). *ISPRS International Journal of Geo-Information*. **2** (3): 869–887. Bibcode:2013IJGI....2..869M (<https://ui.adsabs.harvard.edu/abs/2013IJGI....2..869M>). doi:10.3390/ijgi2030869 (<https://doi.org/10.3390/ijgi2030869>).
- Pontius, Jr, Robert Gilmore; Parmentier, Benoit (2014). "Recommendations for using the Relative Operating Characteristic (ROC)" (<https://www.researchgate.net/publication/263195341>). *Landscape Ecology*. **29** (3): 367–382. doi:10.1007/s10980-013-9984-8 (<https://doi.org/10.1007/s10980-013-9984-8>). S2CID 15924380 (<https://api.semanticscholar.org/CorpusID:15924380>).
- Pontius, Jr, Robert Gilmore; Pacheco, Pablo (2004). "Calibration and validation of a model of forest disturbance in the Western Ghats, India 1920–1990" (<https://www.researchgate.net/publication/226020087>). *GeoJournal*. **61** (4): 325–334. doi:10.1007/s10708-004-5049-5 (<https://doi.org/10.1007/s10708-004-5049-5>). S2CID 155073463 (<https://api.semanticscholar.org/CorpusID:155073463>).
- Pontius, Jr, Robert Gilmore; Batchu, Kiran (2003). "Using the relative operating characteristic to quantify certainty in prediction of location of land cover change in India". *Transactions in GIS*. **7** (4): 467–484. doi:10.1111/1467-9671.00159 (<https://doi.org/10.1111/1467-9671.00159>). S2CID 14452746 (<https://api.semanticscholar.org/CorpusID:14452746>).
- Pontius, Jr, Robert Gilmore; Schneider, Laura (2001). "Land-use change model validation by a ROC method for the Ipswich watershed, Massachusetts, USA" (<https://www.academia.edu/19474075>). *Agriculture, Ecosystems & Environment*. **85** (1–3): 239–248. doi:10.1016/S0167-8809(01)00187-6 ([https://doi.org/10.1016/S0167-8809\(01\)00187-6](https://doi.org/10.1016/S0167-8809(01)00187-6)).
- Stephan, Carsten; Wesseling, Sebastian; Schink, Tania; Jung, Klaus (2003). "Comparison of Eight Computer Programs for Receiver-Operating Characteristic Analysis" (<https://doi.org/10.1373/49.3.433>). *Clinical Chemistry*. **49** (3): 433–439. doi:10.1373/49.3.433 (<https://doi.org/10.1373/49.3.433>). PMID 12600955 (<https://pubmed.ncbi.nlm.nih.gov/12600955>).
- Swets, John A.; Dawes, Robyn M.; and Monahan, John (2000); *Better Decisions through Science*, Scientific American, October, pp. 82–87
- Zou, Kelly H.; O'Malley, A. James; Mauri, Laura (2007). "Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models" (<https://doi.org/10.1161/circulationaha.105.594929>). *Circulation*. **115** (5): 654–7. doi:10.1161/circulationaha.105.594929 (<https://doi.org/10.1161/circulationaha.105.594929>). PMID 17283280 (<https://pubmed.ncbi.nlm.nih.gov/17283280>).
- Zhou, Xiao-Hua; Obuchowski, Nancy A.; McClish, Donna K. (2002). *Statistical Methods in Diagnostic Medicine*. New York, NY: Wiley & Sons. ISBN 978-0-471-34772-9.

---

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Receiver\\_operating\\_characteristic&oldid=976536795](https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=976536795)"

---

This page was last edited on 3 September 2020, at 14:08 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Source: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

# Classification: ROC Curve and AUC

## ROC curve

An **ROC curve (receiver operating characteristic curve)** is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

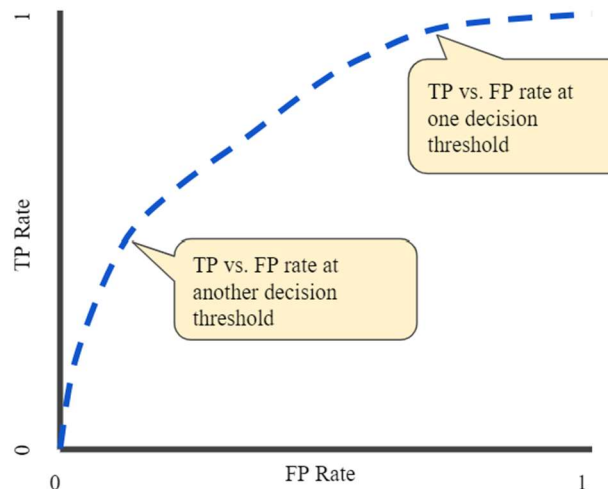
**True Positive Rate (TPR)** is a synonym for recall and is therefore defined as follows:

$$TPR = TP / (TP + FN)$$

**False Positive Rate (FPR)** is defined as follows:

$$FPR = FP / (FP + TN)$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

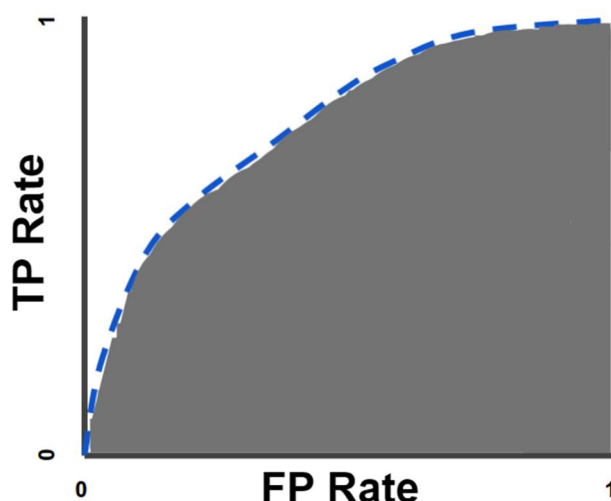


**Figure 4. TP vs. FP rate at different classification thresholds.**

To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

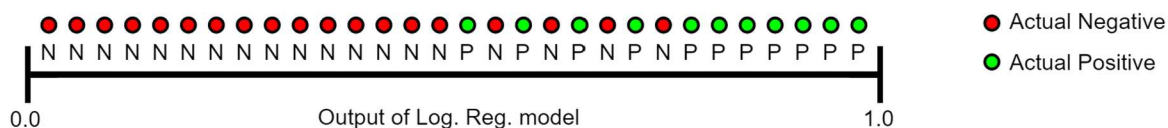
## AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).



**Figure 5. AUC (Area under the ROC Curve).**

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:



**Figure 6. Predictions ranked in ascending order of logistic regression score.**

AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.

- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see the [Google Developers Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Source: <https://sanchom.wordpress.com/2011/09/01/precision-recall/>

# It's a bird... it's a plane... it... depends on your classifier's threshold

Sancho McCann [Uncategorized](#) September 1, 2011

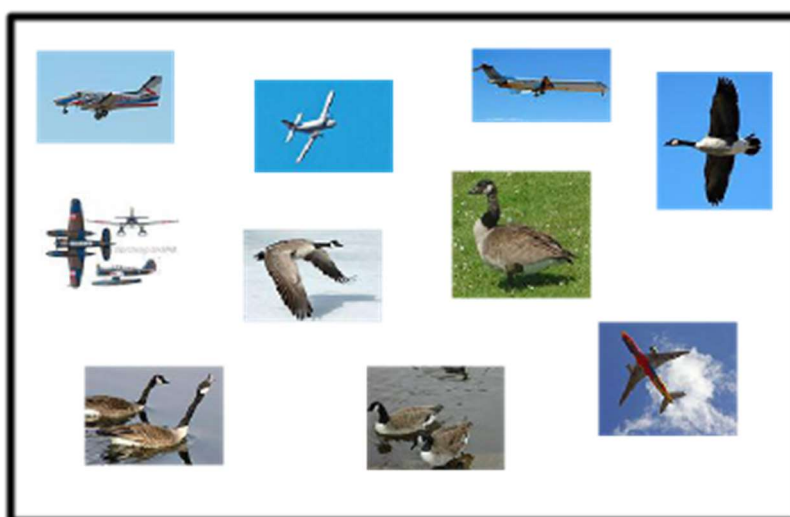
Evaluation of an information retrieval system (a search engine, for example) generally focuses on two things:

1. How relevant are the retrieved results? (precision)
2. Did the system retrieve many of the truly relevant documents? (recall)

For those that aren't familiar, I'll explain what precision and recall are, and for those that are familiar, I'll explain some of the confusion in the literature when comparing precision-recall curves.

## Geese and airplanes

Suppose you have an image collection consisting of airplanes and geese.



You want your system to retrieve all the airplane images and none of the goose images.

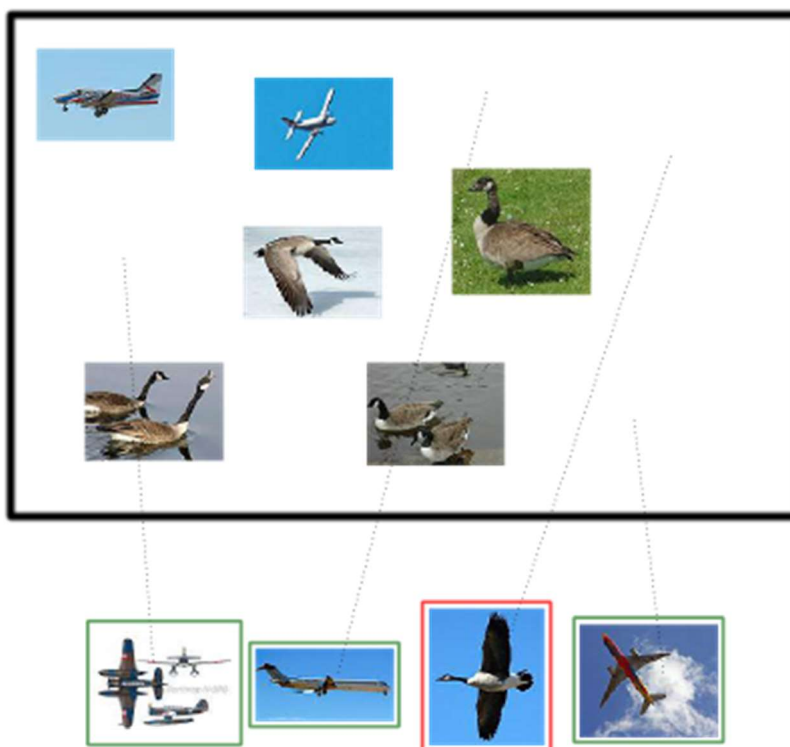
Given a set of images that your system retrieves from this collection, we can define four accuracy counts:

**True positives:** Airplane images that your system correctly retrieved

**True negatives:** Goose images that your system correctly did not retrieve

**False positives:** Geese images that your system incorrectly retrieved, believing them to be airplanes

**False negatives:** Airplane images that your system did incorrectly did not retrieve, believing them to be geese



In this example retrieval, there are three true positives and one false positive.

Using the terms I just defined, in this example retrieval, there are three true positives and one false positive. How many false negatives are there? How many true negatives are there?

There are two false negatives (the airplanes that the system failed to retrieve) and four true negatives (the geese that the system did not retrieve).

## Precision and recall

Now, you'll be able to understand more exactly what *precision* and *recall* are.

Precision is the percentage true positives in the retrieved results. That is:

$$\text{precision} = \frac{tp}{tp + fp} = \frac{tp}{n}$$

where  $n$  is equal to the total number of images retrieved ( $tp + fp$ ).

Recall is the percentage of the airplanes that the system retrieves. That is:

$$\text{recall} = \frac{tp}{tp + fn}$$

In our example above, with 3 true positives, 1 false positive, 4 true negatives, and 2 false negatives, precision = 0.75, and recall = 0.6.

75% of the retrieved results were airplanes, and 60% of the airplanes were retrieved.

## Adjusting the threshold

What if we're not happy with that performance? We could ask the system to return more examples. This would be done by relaxing our threshold of what we want our system to consider as an airplane. We could also ask our system to be more strict, and return fewer examples. In our example so far, the system retrieved four examples. That corresponds to a particular threshold (shown below by a blue line). The system retrieved the examples that appeared more airplane-like than that threshold.



*This is a hypothetical ordering that our airplane retrieval system could give to the images in our collection. More airplane-like are at the top of the list. The blue line is the threshold that gave our example retrieval.*

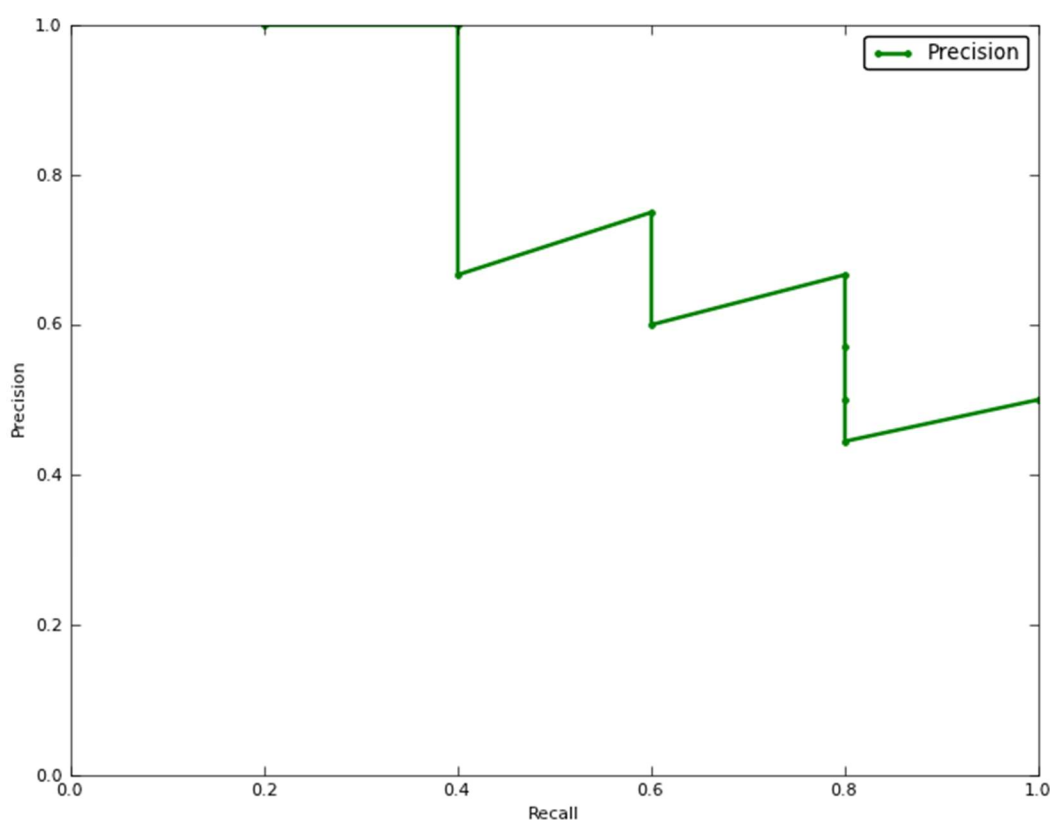
We can move that threshold up and down to get a different set of retrieved documents. At each position of the threshold, we would get a different precision and recall value. Specifically, if we retrieved only the top example, precision would be 100% and recall would be 20%. If we retrieved the top

two examples, precision would still be 100%, and recall will have gone up to 40%. The following chart gives precision and recall for the above hypothetical ordering at all the possible thresholds.

Retrieval cutoff	Precision	Recall
Top 1 image	100%	20%
Top 2 images	100%	40%
Top 3 images	66%	40%
Top 4 images	75%	60%
Top 5 images	60%	60%
Top 6 images	66%	80%
Top 7 images	57%	80%
Top 8 images	50%	80%
Top 9 images	44%	80%
Top 10 images	50%	100%

## Precision-recall curves

A good way to characterize the performance of a classifier is to look at how precision and recall change as you change the threshold. A good classifier will be good at ranking actual airplane images near the top of the list, and be able to retrieve a lot of airplane images before retrieving any geese: its precision will stay high as recall increases. A poor classifier will have to take a large hit in precision to get higher recall. Usually, a publication will present a precision-recall curve to show how this tradeoff looks for their classifier. This is a plot of precision  $p$  as a function of recall  $r$ .



The precision-recall curve for our example airplane classifier. It can achieve 40% recall without sacrificing any precision, but to get 100% recall, its precision drops to 50%.

## Average precision

Rather than comparing curves, it's sometimes useful to have a single number that characterizes the performance of a classifier. A common metric is the *average precision*. This can actually mean one of several things.

### Average precision

Strictly, the average precision is precision averaged across all values of recall between 0 and 1:

$$\int_0^1 p(r) dr$$

That's equal to taking the area under the curve. In practice, the integral is closely approximated by a sum over the precisions at every possible threshold value, multiplied by the change in recall:

$$\sum_{k=1}^N P(k) \Delta r(k)$$

where  $N$  is the total number of images in the collection,  $P(k)$  is the precision at a cutoff of  $k$  images, and  $\Delta r(k)$  is the change in recall that happened between cutoff  $k-1$  and cutoff  $k$ .

In our example, this is  $(1 * 0.2) + (1 * 0.2) + (0.66 * 0) + (0.75 * 0.2) + (0.6 * 0) + (0.66 * 0.2) + (0.57 * 0) + (0.5 * 0) + (0.44 * 0) + (0.5 * 0.2) = 0.782$ .

Notice that the points at which the recall doesn't change don't contribute to this sum (in the graph, these points are on the vertical sections of the plot, where it's dropping straight down). This makes sense, because since we're computing the area under the curve, those sections of the curve aren't adding any area.

### Interpolated average precision

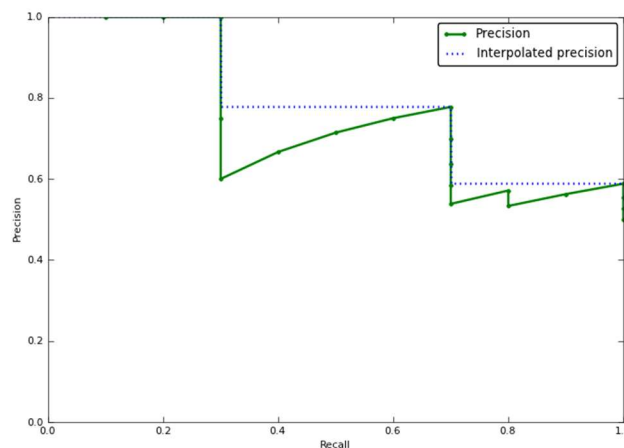
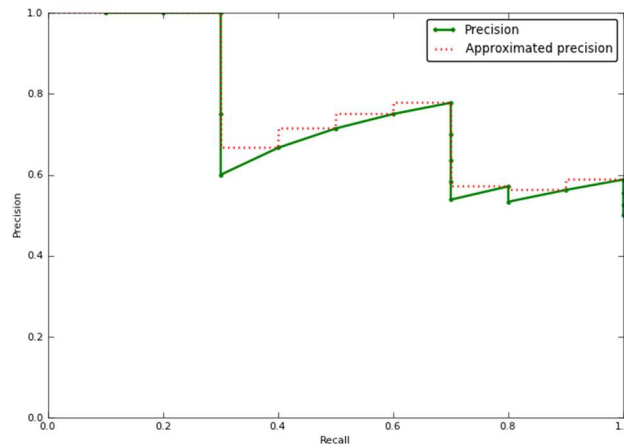
Some authors choose an alternate approximation that is called the *interpolated average precision*. Often, they still call it average precision. Instead of using  $P(k)$ , the precision at a retrieval cutoff of  $k$  images, the interpolated average precision uses:

$$\max_{\tilde{k} \geq k} P(\tilde{k})$$

In other words, instead of using the precision that was actually observed at cutoff  $k$ , the interpolated average precision uses the maximum precision observed across all cutoffs with higher recall. The full equation for computing the interpolated average precision is:

$$\sum_{k=1}^N \max_{\tilde{k} \geq k} P(\tilde{k}) \Delta r(k)$$

Visually, here's how the interpolated average precision compares to the approximated average precision (to show a more interesting plot, this one isn't from the earlier example):



The approximated average precision closely hugs the actually observed curve. The interpolated average precision over estimates the precision at many points and produces a higher average precision value than the approximated average precision.

Further, there are variations on where to take the samples when computing the interpolated average precision. Some take samples at a fixed 11 points from 0 to 1:  $\{0, 0.1, 0.2, \dots, 0.9, 1.0\}$ . This is called the 11-point interpolated average precision. Others sample at every  $k$  where the recall changes.

## Confusion

Some important publications use the interpolated average precision as their metric and still call it average precision. For example, the PASCAL Visual Objects Challenge has used this as their evaluation metric since 2007. I don't think their justification is strong. They say, "the intention in interpolating the precision/recall curve in this way is to reduce the impact of the "wiggles" in the precision/recall curve". Regardless, everyone compares against each other on this metric, so within the competition, this is not an issue. However, the rest of us need to be careful when comparing "average precision" values against other published results. Are we using the VOC's interpolated average precision, while previous work had used the non-interpolated average precision? This would incorrectly show improvement of a new method when compared to the previous work.

## Summary

Precision and recall are useful metrics for evaluating the performance of a classifier.

Precision and recall vary with the strictness of your classifier's threshold.

There are several ways to summarize the precision-recall curve with a single number called average precision; be sure you're using the same metric as the previous work that you're comparing with.