



## Using R and Python in the Teradata Database



## Table of Contents

- 2 Abstract
- 2 Overview
- 3 Introduction
- 4 Analytic Processing in Teradata
- 5 Rules of Engagement
- 5 Use-Cases
- 8 Conclusion
- 8 About the Author

## Abstract

The ever-growing field of advanced analytics presents Teradata® users with many solutions for their mathematical and statistical tasks. In this field, R and Python are two of the most popular and influential open source scripting languages. Since the release of Teradata Database version 15.00, it has been possible to execute these scripting languages directly within the RDBMS, providing the performance and scalability required for analytics against exponentially growing data volumes, such as that generated by the Internet of Things (IoT). This important technology allows for quick exploitation of these key open source technologies by a modern data warehouse, providing the organization with a platform for the Analytics of Things (AoT) and similarly high scale enterprise analytic requirements. The techniques described in this white paper allow both cost savings and access to programming skills that otherwise would not be used within and against data stored in the Teradata Database.

## Overview

It is no secret that both R and Python have become two of the most popular languages for data analytics. Both have their supporters and opponents—while Python is often praised for being a general-purpose language with an easy-to-understand syntax and applicability to a variety of engineering disciplines, R was specifically developed with statisticians in mind perhaps giving it the edge to most of the data science community.

What may be a secret, however, is that both of these open source scripting languages can be used to process data directly within the Teradata Database. With the release of Teradata Database 15.0, data scientists have been able to write their R and Python scripts following several easy to use conventions, so that the analytic processing happens in Teradata. This has tremendous processing and performance benefits, driven by two key attributes:

1. R and Python code is able to leverage Teradata's massively parallel platform (MPP) for performance and scalability.
2. Data resident in the Teradata Database does not have to be moved to another analytic processing platform within the enterprise.

Why is this important? In the past, data scientists have used R or Python on their laptops or an analytic server with memory and processor speed sufficient for the data sets they were analyzing. Today, these same R or Python scripts will not be able to process data volumes that are growing exponentially. If, in the best case scenario these scripts still run, processing times become unbearably slow. By providing a highly scalable MPP platform, these same analytics can be performed against this same data in a timely manner, freeing the data scientists to do more of what they do best—analyze the data.

Scalable performance has a profound effect on analyst productivity and accuracy. If it takes 2-5 minutes to run an analysis, the expert willingly iterates through several parameter refinements and resubmits the analysis. When the processing requires 2-5 hours, their willingness to make accuracy improvements fades away. Along with it, the spirit of agile exploration also is lost. Furthermore, by bringing modern open source technologies inside the Teradata Database, organizations have access to programming skills that provide cost savings and an ability to hire the best young talent entering the workforce today.

This white paper explores several use cases for utilizing the R and Python capabilities available within Teradata today. At Teradata Labs, we believe that both tools will be used within the same corporation depending on skills, preferences, and the task at hand. Teradata has chosen a generic\* mechanism for processing both languages giving you the freedom and flexibility to utilize either, or both, of these powerful languages for data science.

## Introduction

The Teradata Database was built from the ground up for analytics. As the database has evolved over the last several decades, a number of facilities have been added to the core database engine that enables Teradata Database users to bring analytics to the data stored in the database. The design of the shared nothing architecture\*\* of the Teradata Database allows much more than the simple storage and reporting of voluminous detailed data

records. Simply put, the Teradata Database allows the execution of analytical tasks directly on the data, in parallel by using in-database analytics, a capability Teradata Labs pioneered in the late 1990s.

Moving the analytics to the data is a desirable feature for R and Python users who want the ability to process large amounts of data with scripts written in these external languages. In-database analytics can be particularly useful for R and Python users, which without Teradata, need to export large volumes of data to analytic servers. Because these languages can be memory bound, preventing large data volume processing or building applications that can scale, they become unsuitable for use with data and projects where memory or processing demands exceed the capabilities of their computing platform.

Teradata Labs has thoroughly embraced in-database analytics over the last 15-20 years from partnerships with companies such as SAS® and SPSS®, to the internal development of the Teradata Warehouse Miner (TWM) product family. Teradata also recognized the popularity that R was gaining in the late 2000's by creating a package known as `teradataR`. This package, which is now an open source project residing on [Github](#) performs SQL generation to push the processing of popular R functions into the database.

Teradata Labs continued in-database analytic investment by providing infrastructure known as table operators in Teradata Database 14.10. This foundational component extended the capabilities of user defined functions (UDFs)—which allow users to place a custom function in the FROM clause of a SQL SELECT statement—in several important ways:

1. Table operators are object oriented and allow arbitrary inputs and outputs defined at run-time, instead of fixed inputs and outputs defined at compile-time.
2. Table operators have a simpler interface to iterate over the output rows produced by the function, providing a more natural application development interface than table UDF's have.

\* Although the focus of this white paper is R and Python processing, this generic interface can also process other current (Perl, Ruby, GO, etc.) or future languages that follow the same protocols

\*\* In a shared nothing distributed computing architecture, each processing node is independent and self-sufficient. The nodes share no memory or disk storage, and there is no single point of contention across the system. so that the maximum performance and scalability is achieved.

3. Table operators operate on a stream of rows, buffered for high-performance, eliminating the row at a time processing of table UDF's.
4. Table operators support PARTITION BY and ORDER BY, allowing the development of Map and Reduce style operations in-database.

Since the release of this initial capability, Teradata Labs has developed a set of built-in table operators that are advanced analytic enablers. For example, the CALCMATRIX table operator builds matrices of up to 2048 data elements from an arbitrarily large number of data rows, which can then be processed in certain clustering, regression and data reduction algorithms. Then in Teradata Database 15.0, the SCRIPT table operator was released—this built in table operator is the enabler for the execution of R and Python scripts (amongst others) within the Teradata Database.

## Analytic Processing in Teradata

In Teradata Database's shared nothing MPP architecture, the data is evenly distributed across every unit of parallelism, known as AMPs or Access Module Processors. These are virtual processes, multiple instances of which reside on every node—the physical server or cloud virtual instance on which the Teradata Database is installed and running. In order to enable R and/or Python processing on the Teradata platform, the respective interpreters, base packages and any desired add-on packages need to be installed on every node. When execution occurs, the processing is simultaneously performed on every AMP in the system against the data available to that unit of parallelism. The number of AMPs per node varies from 30 to 45 units of parallelism based on the version of the Intel CPU and overall node performance. Thus a ten node system with 40 AMPs per node will run 400 parallel processes of R or Python.

This is important because each instance of R or Python is operating independently, with no inter-process communication across nodes or between the AMPs within a node. The implication here is that the R or Python program must be cognizant of the distribution of data that it is operating against. This is a new paradigm for data scientists who are used to having all their data available in a singular memory address space.

Despite this new processing paradigm, there are a wide variety of use cases that can be addressed. In order to explain them all, we introduce the following processing nomenclature:

- **Row-independent processing (RI)** – The result of the analytic depends only on the input from individual data rows on a single AMP. Examples of RI analytic operations are model scoring, text field parsing and simple scalars (i.e.  $\log(x)$ ).
- **Partition-independent processing (PI)** – The result of the analytic depends on the input from individual data partitions on a single AMP. An example of a PI analytic operation is model fitting for a given region, time period, product or location.
- **System-wide processing** – The result of the analytic is based upon the entire input table which is evenly spread across every AMP in the system. This is the situation where additional design or programming may be needed. System wide processing is required for an operation such as a global sales average, or an attrition model for the entire customer base.

Out of the box, R and Python scripts can be executed in Teradata through the RI and PI processing models. For system-wide style processing, the data scientist must construct a master processing level to combine and appropriately process the partial results returned from every AMP process. This can be done in either a MapReduce style by nesting multiple calls to the SCRIPT table operator or by embedding calls to the SCRIPT table operator within a C++ or Java external stored procedure (XSP). In either case, the results are aggregated across all AMPs and processed further to produce a meaningful final answer.

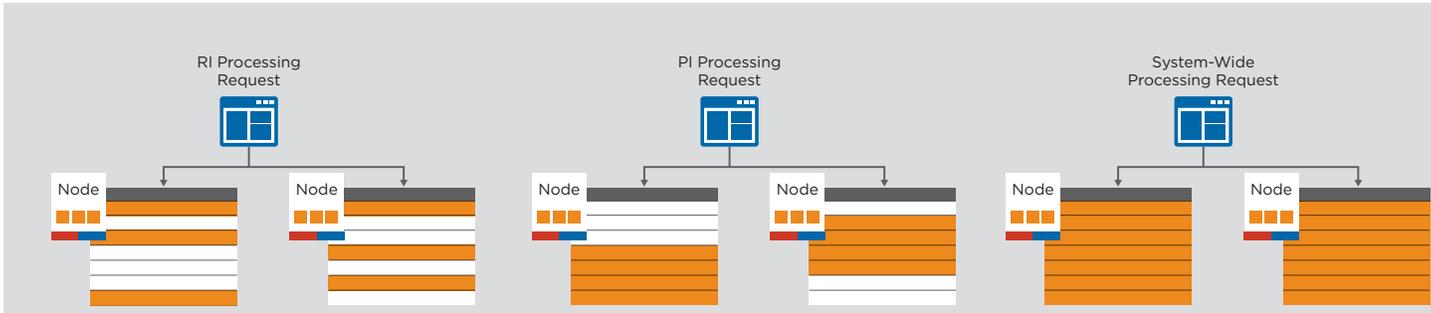


Figure 1. Analytic Processing Paradigms.

## Rules of Engagement

In order for an R or Python script to be processed in Teradata, there are certain simple rules that must be followed. First, the script must be “installed” or registered to the database. The Teradata Database provides a very simple one line SQL command that performs this registration. Second, the R or Python script must read data from the database through the Standard Input Stream, commonly referred to as **STDIN**, and write to the database through the Standard Output Stream or **STDOUT**. For the first processing model described above (RI), these are the only two rules.

The second processing model (PI) requires an additional rule be followed. These programs assume that each data partition will be hashed appropriately to every AMP. Teradata provides indexing mechanisms such as partitioned primary indexes that guarantee these hashing schemes are applied to the tables being operated on. When writing the SQL statement to source the data for use within the script, a **PARTITION BY** clause can also be used to feed the R or Python script the data it needs to process each group of data.

For the last processing model, additional programming will be required, depending upon the use case. We will explore a system-wide processing use case in subsequent sections of this white paper.

## Use-Cases

### Scoring in R

Analytic model scoring is perhaps the most common example of RI processing. Consider for a moment that you are an insurance carrier who has been using R on a large Windows server for quite some time. You have developed a generalized linear model (GLM) to predict propensity to purchase an upgraded policy based upon demographic, historical policy and claim data resident in the database. You’ve been asked to score the entire customer base with your model, but the Windows server is running out of memory when you apply it to the entire corpus.



Since the data resides in the Teradata Database, the **SCRIPT** table operator is used to apply the model to all customers and give them a propensity to buy score for this new policy. The scoring script is used to read the data from **STDIN** and write it to **STDOUT**, and the script is registered to the Teradata Database. The script reads model parameters from the R model object built on the Windows system, applies those parameters to the entire population and creates a table of customer propensity to buy scores in several seconds.

## Simulation in Python

Simulation is a different example of RI processing. Simulation is a flexible methodology used to analyze the behavior of a proposed business activity such as a new product launch. By performing simulations and analyzing the results in the database, we can estimate how a proposed new product would perform in the marketplace and the effects on the present product portfolio.

Another good example of a use case for simulation is supply chain. Suppose a retailer is having significant restocking problems, and needs to optimize their inventory levels because of fluctuating demand at some of the retail stores.

The analytics team have written a fairly complex program in Python, utilizing the Python package “SimPy” In this scenario, various inventory stock levels are studied for certain products based upon store location, population demographics, seasonality and sales to determine how best to provide stock to the stores. The problem is the Linux server runs out of resources and crashes before the study can be completed.



In order to simulate a process over multiple iterations in Teradata Database, you provide an input table with rows of random seeds. Each row provides a seed for an individual iteration; hence the results from all AMPs correspond to the simulation output from independent iterations. Different random seeds across iterations serve to define different random inventory stock levels. Simulation results are then aggregated inside the parallel database to complete the study. Following the processing rules for the SCRIPT table operator, the Python simulation program completes in several minutes, instead of never or days.

For some, it may seem unusual to utilize a relational database to simulate data. However, considering the compute intensive nature of simulation uses cases, it is a very powerful option to leverage Teradata Database’s R and Python processing capabilities. The additional analysis that can be done with historical data in comparison to the simulated data may also provide insights that were never thought of before.

## Clustering in Python

Clustering lends itself to the PI processing paradigm when needing to group data together. For example, a telecommunications company that has acquired a new region often needs to establish a new cellular network by putting towers in a particular location. This necessitates building a clustering model for the users in this region to ensure optimal signal strength for each.

Using iPython on a laptop, along with the PyCluster package, a K-Means model for the user population within a postal code area in the new region can be built. Utilizing call detail records exported from the data warehouse onto a laptop reveals several clusters that appear optimal for one particular geographical area. However, when attempting to apply it all the postal codes in the region, the Python script crashes the system.



Typically the call detail records stored in the data warehouse have been partitioned by the postal code, making this use-case perfect for the SCRIPT table operator. By modifying the Python script to read the table directly through STDIN as opposed to a flat file on a laptop, the power of the MPP platform can be exploited to determine clustering schemes for all of postal codes within the new region in a matter of minutes. Additionally, the cluster scores have been built directly in the database so that applications used by the telco’s field engineers have immediate access to the data for planning and other purposes.

## Multiple Model Fitting and Scoring in R

Any type of modeling algorithm available in R or Python is able to leverage PI processing in Teradata Database to build multiple models. By ensuring that the data selected has been built with a partitioning scheme that matches the analytic models being built, the SCRIPT table operator can be leveraged. For example, building a model for each product a retailer sells requires the data be partitioned by a product identifier. The resulting model can be stored in the database in a temporary table and then read by a scoring routine to apply the model to other data in the Teradata Database.

Consider, for example, a financial institution with an attrition problem. Upper management has asked the marketing team to put together attractive packages of various accounts and services that the bank offers customers who are at risk of moving to another bank. The goal is to develop an R program that utilizes the random forest R package to build a classification model to determine which products have a higher probability of cross-selling when the customer already has another banking service. However, when executing the random forest R package to build a model for each of the 1000s of existing products/services the bank sells, the program hangs and never completes. This is another example where in-database parallelism is necessary.



Applying decision tree analysis in Teradata Database is a two-step approach. First, a model fitting script that reads the training data and applies the random forest algorithm to it is executed. The fitted model details are sent directly to the Teradata Database by serializing the results and storing them in a table as a character large object (CLOB). Then, a second script reads the serialized model, transforms it back to its original format so that the predict() R function can score the entire customer population. Both steps in the process complete in under an hour and you are a hero in the marketing department.

## Modeling the Population

While R and Python scripts typically require little to no change to operate within the RI or PI paradigms, the same is not true of system-wide processing. When dealing with the entire corpus, the SCRIPT table operator can be used in a MapReduce way by nesting multiple calls. At the inner—or local—level, an R or Python script produces partial results from each AMP. Then in the outer—or global—level, a different script receives the output from the local level, and aggregation operations on this partial output produce a global result.

It can be difficult for a data scientist to think in terms of the MapReduce paradigm, where one R or Python script is your mapper that produces partial results, and the other a reducer that aggregates the partial results into a final answer. However, one of the very interesting things you can do is build your R or Python script so that it reads directly from a correlation or covariance matrix. There are many classes of analytics which build a matrix as the first step in its processing—the most popular of which are multivariate linear regression and data reduction through factor analysis. For these types of analytics, the SCRIPT table operator can work in a nested MapReduce manner with Teradata's built-in CALCMATRIX table operator.

For example, an IT department wants to forecast future system performance and capacity. In particular, they



want the answer to the following question—if the workload on the data warehouse grows by 10% each quarter, when will we need to grow the system? This requires a forecast over many years to pinpoint when the system runs out of compute capacity. And for that same exact reason, the Teradata Database is needed for its in-database processing power. We call this “Analytics on Analytics.”

One approach to this problem is to build a regression model. The first step in the process is to calculate the sum of squares and cross-products matrix from the system's utilization data. For this task, the CALCMATRIX table operator is employed in a nested call within a SCRIPT table operator call which will build a regression model from the R script. The CALCMATRIX table operator easily and quickly, creates the matrix which is streamlined as input to the outer call of the nested query to the SCRIPT table operator. The R script then takes over the analysis with the computed matrix as input, and produces a list of the regression estimates for the intercept and variable coefficients. Once the model statistics are satisfactory, the regression model is applied for the forecast. Imagine using simple nested SQL statements to build and score models against Teradata Database's own system logs in a matter of minutes.

## Conclusion

Teradata truly understands the power of bringing analytics to the data for performance and scalability. By extending the capabilities of the platform to utilize the two most popular languages for data science, Teradata Labs has provided R and Python programmers an elegant way to run these languages in parallel without having to move huge amounts of data to another server. They can move the processing to the data instead. By keeping the data in place, and utilizing the compute power of the MPP platform, R and Python programmers can quickly run their analytics at any scale. This gives them more time to discover new, incredibly valuable insights, for the business.

## About the Author

Tim Miller has been in a wide variety of R&D roles at Teradata Labs over his 29 year career. He has been involved in all aspects of enterprise software development, from software architecture to quality assurance. Tim has developed software in many domains—the last 20 years dedicated to predictive analytics. He is one of two principals in the development of the first commercial in-database data mining system, Teradata Warehouse Miner. Recently, as a member of Teradata's Partner Integration Lab, he consulted with new and emerging Teradata ISV partners to integrate and optimize their advanced analytics products with Teradata's platform family. Today, Tim is part of Teradata's Advanced Analytics Consulting Practice working closely with customers to optimize their analytic environments.

This document, which includes the information contained herein, (i) is the exclusive property of Teradata Corporation; (ii) constitutes Teradata confidential information; (iii) may not be disclosed by you to third parties; (iv) may only be used by you for the exclusive purpose of facilitating your internal Teradata-authorized use of the Teradata product(s) described in this document to the extent that you have separately acquired a written license from Teradata for such product(s); and (v) is provided to you solely on an "AS-IS" basis. In no case will you cause this document or its contents to be disseminated to any third party, reproduced or copied by any means (in whole or in part) without Teradata's prior written consent. Any copy of this document, or portion thereof, must include this notice, and all other restrictive legends appearing in this document. Note that any product, process or technology described in this document may be the subject of other intellectual property rights reserved by Teradata and are not licensed hereunder. No license rights will be implied. Use, duplication, or disclosure by the United States government is subject to the restrictions set forth in DFARS 252.227-7013(c)(1)(ii) and FAR 52.227-19. Other brand and product names used herein are for identification purposes only and may be trademarks of their respective companies.

This white paper has a detailed counterpart known as an "Orange Book." The "R and Python Analytics with the SCRIPT Table Operator Introductory Guide" includes detailed examples and corresponding data sets for the types of use cases described above. It also includes in-depth installation instructions to enable R and Python scripting on your Teradata platform. Orange books are available at [Teradata @ Your Service](#).

Unified Data Architecture is a trademark, and Teradata and the Teradata logo are registered trademarks of Teradata Corporation and/or its affiliates in the U.S. or worldwide. Linux is a registered trademark of Linus Torvalds. Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Python is a registered trademark of the Python Software Foundation. Red Hat is a registered trademark of Red Hat, Inc. SAS is a registered trademark of SAS Institute Inc. SLES is a registered trademark of Novell, Inc. SPSS is a registered trademark of IBM Corporation. SUSE is a registered trademark of SUSE, LLC. Teradata continually improves products as new technologies and components become available. Teradata, therefore, reserves the right to change specifications without prior notice. All features, functions, and operations described herein may not be marketed in all parts of the world. Consult your Teradata representative or [Teradata.com](#) for more information.

Copyright © 2016 by Teradata Corporation All Rights Reserved. Produced in U.S.A.

04.16 EB9386



TERADATA