

Contents

Overview

[What is SQL Server Machine Learning Services?](#)

[In-Database analytics](#)

[Standalone server](#)

[New features](#)

[Features by edition](#)

Architecture

[R](#)

[R interoperability](#)

[Components that support R integration](#)

[Security for R](#)

[Python](#)

[Python in Machine Learning Services](#)

[Python interoperability](#)

[Components to support Python](#)

[Python security](#)

Install

[SQL Server 2017](#)

[In-Database analytics](#)

[Standalone server](#)

[SQL Server 2016](#)

[R Services \(In-Database\)](#)

[R Server \(Standalone\)](#)

[Pre-trained models](#)

[Command-prompt setup](#)

[Offline setup \(no internet\)](#)

[Upgrade R and Python](#)

[Set up R tools](#)

[Set up Python tools](#)

Quickstarts

R

- [Hello World in R and SQL](#)
- [Handle inputs and outputs](#)
- [Handle data types and objects](#)
- [Create a predictive model](#)
- [Predict and plot from model](#)

Tutorials

R

- [Learn in-database analytics](#)
 - [1 - Get data and scripts](#)
 - [2 - Set up the environment](#)
 - [3 - Visualize data](#)
 - [4 - Create data features](#)
 - [5 - Train and save to SQL](#)
 - [6 - Predict outcomes](#)

Data science end-to-end walkthrough

- [Prepare data](#)
- [Explore data using SQL](#)
- [Summarize data using R](#)
- [Create graphs and plots](#)
- [Create data features](#)
- [Build and save the model](#)
- [Deploy and use the model](#)

Deep dive with RevoScaleR

- [Create database and permissions](#)
- [Create data objects using RxSqlServerData](#)
- [Query and modify data](#)
- [Define a compute context](#)
- [Create and run R scripts](#)
- [Visualize data](#)
- [Create models](#)

- Score new data
- Transform data
- Load data into memory using rxImport
- Create a table using rxDataStep
- Perform chunking analysis using rxDataStep
- Analyze data in local compute context
- Move data between SQL Server and XDF file
- Create a simple simulation

Python

- Run Python using T-SQL
- Wrap Python in a stored procedure
- Train and score from a Python model in SQL Server
- Create a model using revoscalepy in a SQL Server compute context
- Learn in-database analytics
 - Get data and scripts
 - Import data
 - Explore and visualize data
 - Create data features
 - Train and save the model
 - Operationalize the model

Samples

Solutions

How To

- Package management
 - Default packages
 - Get package information
 - Install new Python packages
 - Install new R packages
 - Use R package managers
 - Use T-SQL
 - Use RevoScaleR
 - Enable remote R package management

- Synchronize R packages
 - Create a miniCRAN repo
 - Tips for using R packages
- Data exploration and modeling
 - R libraries and data types
 - Python libraries and data types
 - Native scoring
 - Real-time scoring
 - Predictive modeling with R
 - How to perform real-time or native scoring
 - Load R objects using ODBC
 - Converting R Code for Use in Machine Learning Services
 - Creating multiple models using rxExecBy
 - Use data from OLAP cubes in R
 - Create a stored procedure using sqlrutils
- Performance
 - Performance tuning for R - Overview
 - Performance tuning for R - SQL Server configuration)
 - Performance tuning for R - R and data optimization
 - Performance tuning for R - Results
 - Use R code profiling functions
- Administration
 - Configure and manage R
 - Advanced configuration options for Machine Learning Services
 - Security considerations for the R runtime in SQL Server
 - Modify the user account pool for SQL Server Machine Learning Services
 - Add SQLRUserGroup as a database user
 - Deploy and consume models using web services
 - Manage and monitor solutions
 - Resource governance for Machine Learning Services
 - Create a resource pool for machine learning
 - Extended events for Machine Learning Services

- [Extended events for monitoring PREDICT statements](#)
- [DMVs for Machine Learning Services](#)
- [Using R code profiling functions](#)
- [Monitor Machine Learning Services using custom reports in Management Studio](#)
- [Package reference](#)
 - [MicrosoftML in SQL](#)
 - [RevoScaleR in SQL](#)
 - [RevoScaleR function list for SQL Server data](#)
 - [sqlrutils in SQL](#)
 - [olapR in SQL](#)
 - [revoscalepy in SQL](#)
- [Resources](#)
 - [Known issues](#)
 - [Release notes](#)
 - [Set up a virtual machine](#)
 - [Troubleshooting](#)
 - [Data collection](#)
 - [Install and upgrade errors](#)
 - [Launchpad and external script execution errors](#)
 - [R scripting errors](#)
- [Blogs](#)
 - [SQL Server](#)
 - [R Server](#)
 - [Machine Learning](#)
- [Forums](#)
 - [SQL Server](#)
 - [Machine Learning Server](#)

What is SQL Server Machine Learning Services?

5/30/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server Machine Learning Services is an embedded, predictive analytics and data science engine that can execute R and Python code within a SQL Server database as stored procedures, as T-SQL script containing R or Python statements, or as R or Python code containing T-SQL.

The key value proposition of Machine Learning Services is the power of its proprietary packages to deliver advanced analytics at scale, and the ability to bring calculations and processing to where the data resides, eliminating the need to pull data across the network.

There are two options for using machine learning capabilities in SQL Server:

- **SQL Server Machine Learning Services (In-Database)** operates within the database engine instance, where the calculation engine is fully integrated with the database engine. Most installations are this option.
- **SQL Server Machine Learning Server (Standalone)** is Machine Learning Server for Windows that runs independently of the database engine. Although you use SQL Server Setup to install the server, the feature is not instance-aware. Functionally, it is equivalent to the non-SQL-Server [Microsoft Machine Learning Server for Windows](#).

R and Python packages

Support for each language is through proprietary Microsoft packages used for creating and training models of various types, scoring data, and parallel processing using the underlying system resources.

Because the proprietary packages are built on open-source R and Python distributions, script or code that you run in SQL Server can also call base functions and use third-party packages compatible with the language version provided in SQL Server (Python 3.5 and recent versions of R, currently 3.3.3).

R	PYTHON	DESCRIPTION
RevoScaleR	revoscalepy	Functions in these libraries are among the most widely used. Data transformations and manipulation, statistical summarization, visualization, and many forms of modeling and analyses are found in these libraries. Additionally, functions in these libraries automatically distribute workloads across available cores for parallel processing, with the ability to work on chunks of data that are coordinated and managed by the calculation engine.
MicrosoftML	microsoftml	Industry-leading machine learning algorithms for image featurization, classification problems, and more.
olapR	none	Build or execute an MDX query in R script.

R	PYTHON	DESCRIPTION
sqlRUtils	none	Functions for putting R scripts into a T-SQL stored procedure, registering a stored procedure with a database, and running the stored procedure from an R development environment.
mrsdeploy	none	Primarily used on a non-SQL installation of Machine Learning Server, such as the (Standalone) version . Use this package to deploy and host web services, build scale-out topologies with dedicated web and compute nodes, toggle between local and remote sessions, run diagnostics, and more. For an (In-Database) installation, use this package in a client capacity: for example, to access a web service on a remote server dedicated to running just Machine Learning Services workloads.

Portability of your custom R and Python code is addressed through package distribution and interpreters that are built into multiple products. The same packages that ship in SQL Server are also available in several other Microsoft products and services, including a non-SQL version called [Microsoft Machine Learning Server](#). Free clients that include our R and Python interpreters include [Microsoft R Client](#) and the [Python libraries](#).

Packages and interpreters are also available on several [Azure virtual machines](#), Azure Machine Learning, and Azure services like [HDInsight](#).

Use cases

In-database analytics

Developers and analysts often have code running on top of a local SQL Server instance. If you have SQL Server and an IDE such as [Visual Studio with R](#) or [Visual Studio with Python](#) on the same computer, you can build, train, and test models locally in either language. Local access simplifies package management: as an admin, you can load additional third-party packages using built-in permissions for that role.

The most common approach for in-database analytics is to use [sp_execute_external_script](#) to run R or Python script. The tutorials listed in [Next steps](#) will get you started.

Client-server configurations

From any client workstation that has an IDE, install [Microsoft R Client](#) or the [Python libraries](#), and then write code that pushes execution (referred to as a *remote compute context*) to data and operations to a remote SQL Server.

Similarly, if you are using the Developer edition, you can build solutions on a client workstation using the same libraries and interpreters, and then deploy production code on SQL Server Machine Learning Services (In-Database).

Version history

SQL Server 2017 Machine Learning Services is the next generation of SQL Server 2016 R Services, enhanced to include Python. The following table is a complete list of all versions, from inception to the current release.

Product Name	Engine Version	Release Date
SQL Server 2017 Machine Learning Services (In-Database)	R Server 9.2.1 Python Server 9.2	October 2017
SQL Server 2017 Machine Learning Server (Standalone)	R Server 9.2.1 Python Server 9.2	October 2017
SQL Server 2016 R Services (In-Database)	R Server 9.1	July 2017
SQL Server 2016 R Server (Standalone)	R Server 9.1	July 2017

Documentation for each version

Recent releases of SQL Server documentation are version-agnostic. For SQL Server Machine Learning Services, Python is only available in 2017 and later, while R support is in all versions. Unless noted otherwise, you can assume R documentation applies to both 2016 and 2017 versions.

Related machine learning products

- [Provision an Azure Virtual Machine](#)

The Azure marketplace includes multiple virtual machine images that include Machine Learning Server or R Server. Creating a virtual machine in Microsoft Azure is the fastest way to get to development and deployment of predictive models. Images come with features for scaling and sharing already configured, which makes it easier to embed analytics inside applications and to integrate with backend systems.

- [Data Science Virtual Machine](#)

The latest version of the Data Science Virtual machine includes Machine Learning Server, SQL Server, plus an array of the most popular tools for machine learning, all preinstalled and tested. Create Jupyter notebooks, develop solutions in Julia, and use GPU-enabled deep learning libraries like MXNet, CNTK, and TensorFlow.

Next steps

Step 1: Install and configure the software.

- [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#)

Step 2: Get started with code using either one of these tutorials:

- [Tutorial: Run Python in T-SQL](#)
- [Tutorial: Run R in T-SQL](#)

Step 3: Add your favorite R and Python packages and use them together with packages provided by Microsoft

- [R Package management for SQL Server](#)

SQL Server Machine Learning and R Services (In-Database)

7/20/2018 • 7 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

An in-database installation of machine learning operates within the context of a SQL Server database engine instance, providing R and Python external script support for resident data in your SQL Server instance. Because machine learning is integrated with SQL Server, you can keep analytics close to the data and eliminate the costs and security risks associated with data movement.

Because the database engine is multi-instance, you can install more than one instance of in-database analytics, or even older and newer versions side-by-side. Choices include either [SQL Server 2017 Machine Learning Services \(In-Database\)](#) with R and Python, or [SQL Server 2016 R Services \(In-Database\)](#) with just R.

Machine learning components can also be installed as instance-agnostic [standalone servers](#). Generally, we recommend that you treat (Standalone) and (In-Database) installations as mutually exclusive to avoid resource contention, but if you have sufficient resources, there are no prohibitions against installing them both on the same physical computer.

Choosing between in-database and standalone analytics

Understanding your development requirements can help you choose between (In-Database) and (Standalone) approaches. A standalone server is simpler to configure and manage if you want maximum flexibility in how it is used, or if you want to also connect to a variety of data sources outside of SQL Server.

In-database analytics are designed for deep integration with data within SQL Server. You can write T-SQL queries that call R or Python functions and execute the script in SQL Server Management Studio or any tool or app used for external or embedded T-SQL. If you need to run R or Python code in SQL Server, either by using stored procedures or by using the SQL Server instance as the [compute context](#), you must install in-database analytics. This option provides maximum data security and integration with SQL Server tools.

Both in-database and standalone servers can alleviate the memory and processing constraints of open-source R and Python. Both options include the same packages and tools, with the ability to load and process large amounts of data on multiple cores and aggregate the results into a single consolidated output. The functions and algorithms are engineered for both scale and utility: delivering predictive analytics, statistical modeling, data visualizations, and leading-edge machine learning algorithms in a commercial server product engineered and supported by Microsoft.

Components of an in-database installation

SQL Server 2016 is R only. SQL Server 2017 supports R and Python. The following table describes the features in each version. With the exception of the SQL Server Launchpad service, this table is identical to the one provided in the [standalone server article](#).

COMPONENT	DESCRIPTION
SQL Server Launchpad service	A service that manages communications between the external R and Python runtimes and the SQL Server instance.

COMPONENT	DESCRIPTION
R packages	<p>RevoScaleR is the primary library for scaleable R with functions for data manipulation, transformation, visualization, and analysis.</p> <p>MicrosoftML (R) adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.</p> <p>mrsdeploy offers web service deployment (in SQL Server 2017 only).</p> <p>olapR is for specifying MDX queries in R.</p>
Microsoft R Open (MRO)	<p>MRO is Microsoft's open-source distribution of R. The package and interpreter are included. Always use the version of MRO bundled in setup.</p>
R tools	<p>R console windows and command prompts are standard tools in an R distribution. Find them at \Program files\Microsoft SQL Server\140\R_SERVER\bin\x64.</p>
R Samples and scripts	<p>Open-source R and RevoScaleR packages include built-in data sets so that you can create and run script using pre-installed data. Look for them at \Program files\Microsoft SQL Server\140\R_SERVER\library\datasets and \library\RevoScaleR.</p>
Python packages	<p>revoscalepy is the primary library for scaleable Python with functions for data manipulation, transformation, visualization, and analysis.</p> <p>microsoftml (Python) adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.</p>
Python tools	<p>The built-in Python command line tool is useful for ad hoc testing and tasks. Find the tool at \Program files\Microsoft SQL Server\140\PYTHON_SERVER\python.exe.</p>
Anaconda	<p>Anaconda is an open-source distribution of Python and essential packages.</p>
Python samples and scripts	<p>As with R, Python includes built-in data sets and scripts. Find the revoscalepy data at \Program files\Microsoft SQL Server\140\PYTHON_SERVER\lib\site-packages\revoscalepy\data\sample-data.</p>
Pre-trained models in R and Python	<p>Pre-trained models are supported and usable on a standalone server, but you cannot install them through SQL Server Setup. The setup program for Microsoft Machine Learning Server provides the models, which you can install free of charge. For more information, see Install pretrained machine learning models on SQL Server.</p>

Get started step-by-step

Start with setup, attach the binaries to your favorite development tool, and write your first script.

Step 1: Install the software

Install either one of these versions:

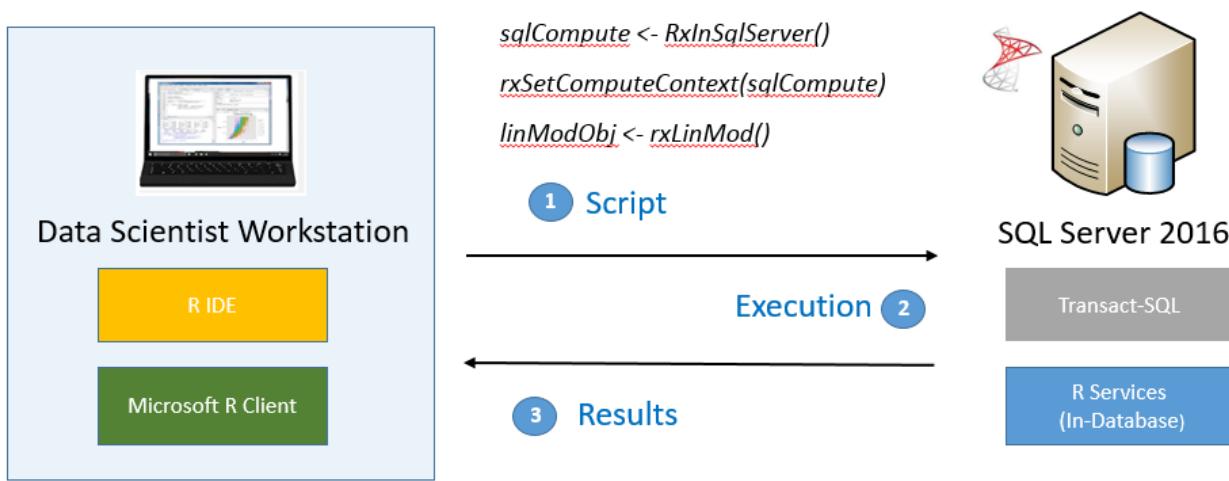
- [SQL Server 2017 Machine Learning Services \(In-Database\)](#)
- [SQL Server 2016 R Services \(In-Database\) - R only](#)

Step 2: Configure a development tool

Configure your development tools to use the Machine Learning Server binaries. For more information about Python, see [Link Python binaries](#). For instructions on how to connect in R Studio, see [Using Different Versions of R](#) and point the tool to C:\Program Files\Microsoft SQL Server\140\R_SERVER\bin\x64. You could also try [R Tools for Visual Studio](#).

Data scientists typically use R or Python on their own laptop or development workstation, to explore data, and build and tune predictive models until a good predictive model is achieved.

With in-database analytics in SQL Server, there is no need to change this process. After installation is complete, you can run R or Python code on SQL Server either locally or remotely:



- **Use the IDE you prefer.** R Services (In-Database) client components provide the data scientist with all the tools needed to experiment and develop. These tools include the R runtime, the Intel math kernel library to boost the performance of standard R operations, and a set of enhanced R packages that support executing R code in SQL Server.
- **Work remotely or locally.** Data scientists can connect to SQL Server and bring the data to the client for local analysis, as usual. However, a better solution is to use the **RevoScaleR** or **revoscalepy** APIs to push computations to the SQL Server computer, avoiding costly and insecure data movement.
- **Embed R or Python scripts in Transact-SQL stored procedures.** When your code is fully optimized, wrap it in a stored procedure to avoid unnecessary data movement and optimize data processing tasks.

Step 3: Write your first script

Call R or Python functions from within T-SQL script:

- [R: Use R code in Transact-SQL](#)
- [R: In-database analytics for SQL developers](#)
- [Python: Run Python using T-SQL](#)
- [Python: In-database analytics for SQL developers](#)

Choose the best language for the task. R is best for statistical computations that are difficult to implement using SQL. For set-based operations over data, leverage the power of SQL Server to achieve maximum performance. Use the in-memory database engine for very fast computations over columns.

Step 4: Optimize your solution

When the model is ready to scale on enterprise data, the data scientist often works with the DBA or SQL

developer to optimize processes such as:

- Feature engineering
- Data ingestion and data transformation
- Scoring

Traditionally data scientists using R have had problems with both performance and scale, especially when using large dataset. That is because the common runtime implementation is single-threaded and can accommodate only those data sets that fit into the available memory on the local computer. Integration with SQL Server Machine Learning Services provides multiple features for better performance, with more data:

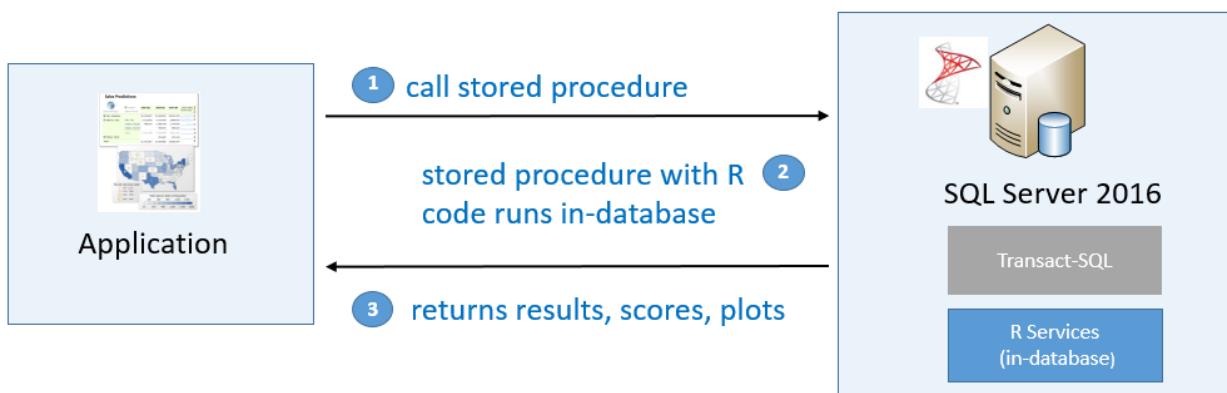
- **RevoScaleR**: This R package contains implementations of some of the most popular R functions, redesigned to provide parallelism and scale. The package also includes functions that further boost performance and scale by pushing computations to the SQL Server computer, which typically has far greater memory and computational power.
- **revoscalepy**. This Python library, available in SQL Server 2017, implements the most popular functions in RevoScaleR, such as remote compute contexts, and many algorithms that support distributed processing.

Resources

- [Performance Case Study](#)
- [R and Data Optimization](#)

Step 5: Deploy and Consume

After the script or model is ready for production use, a database developer might embed the code or model in a stored procedure, so that the saved R or Python code can be called from an application. Storing and running R code from SQL Server has many benefits: you can use the convenient Transact-SQL interface, and all computations take place in the database, avoiding unnecessary data movement.



- **Secure and extensible**. R Services (In-Database) uses a new extensibility architecture that keeps your database engine secure and isolates R and Python sessions. You also have control over the users who can execute scripts, and you can specify which databases can be accessed by code. You can control the amount of resources allocated to the runtime, to prevent massive computations from jeopardizing the overall server performance.
- **Scheduling and auditing**. When external script jobs are run in SQL Server, you can control and audit the data used by data scientists. You can also schedule jobs and author workflows containing external R or Python scripts, just like you would schedule any other T-SQL job or stored procedure.

To take advantages of the resource management and security features in SQL Server, the deployment process might include these tasks:

- Converting your code to a function that can run optimally in a stored procedure
- Setting up security and locking down packages used by a particular task

- Enabling resource governance (requires the Enterprise edition)

Resources

- [Resource Governance for R](#)
- [R Package Management for SQL Server](#)

See also

[SQL Server Machine Learning and R Server \(Standalone\)](#)

SQL Server Machine Learning Server (Standalone) and R Server (Standalone)

7/20/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

A standalone server is an installation of machine learning components, articulated as R and Python features, that run independently of SQL Server database engine instances. You can install a standalone server by itself, with no dependencies on SQL Server. Because a standalone server is independent of SQL Server, configuration and administration tasks and tools are more similar to a non-SQL version of Machine Learning Server, which you can read about in [this article](#).

The objective of a standalone machine learning server is to provide a rich development environment, with distributed and parallel processing of R and Python workloads over small-to-large data sets, using the proprietary packages and calculation engines installed with the server. The R and Python packages on a standalone server are the same as those provided in a SQL Server (In-Database) installation, allowing for code portability and [compute-context switching](#).

The primary reason developers choose a standalone machine learning server is to move beyond the memory and processing constraints of open-source R and Python. Standalone servers can load and process large amounts of data on multiple cores and aggregate the results into a single consolidated output. The functions and algorithms are engineered for both scale and utility: delivering predictive analytics, statistical modeling, data visualizations, and leading-edge machine learning algorithms in a commercial server product engineered and supported by Microsoft.

Generally, we recommend that you treat (Standalone) and (In-Database) installations as mutually exclusive to avoid resource contention, but if you have sufficient resources, there is no prohibition against installing them both on the same physical computer.

You can only have one standalone server on the computer: either [SQL Server 2017 Machine Learning Server \(standalone\)](#) or [SQL Server 2016 R Server \(Standalone\)](#). You must manually uninstall one version before installing a different version.

Components of a standalone server

SQL Server 2016 is R only. SQL Server 2017 supports R and Python. The following table describes the features in each version.

COMPONENT	DESCRIPTION
R packages	<p>RevoScaleR is the primary library for scalable R with functions for data manipulation, transformation, visualization, and analysis.</p> <p>MicrosoftML (R) adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.</p> <p>mrsdeploy offers web service deployment (in SQL Server 2017 only).</p> <p>olapR is for specifying MDX queries in R.</p>

COMPONENT	DESCRIPTION
Microsoft R Open (MRO)	MRO is Microsoft's open-source distribution of R. The package and interpreter are included. Always use the version of MRO bundled in setup.
R tools	R console windows and command prompts are standard tools in an R distribution. Find them at \Program files\Microsoft SQL Server\140\R_SERVER\bin\x64.
R Samples and scripts	Open-source R and RevoScaleR packages include built-in data sets so that you can create and run script using pre-installed data. Look for them at \Program files\Microsoft SQL Server\140\R_SERVER\library\datasets and \library\RevoScaleR.
Python packages	revoscalepy is the primary library for scaleable Python with functions for data manipulation, transformation, visualization, and analysis. microsoftml (Python) adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.
Python tools	The built-in Python command line tool is useful for ad hoc testing and tasks. Find the tool at \Program files\Microsoft SQL Server\140\PYTHON_SERVER\python.exe.
Anaconda	Anaconda is an open-source distribution of Python and essential packages.
Python samples and scripts	As with R, Python includes built-in data sets and scripts. Find the revoscalepy data at \Program files\Microsoft SQL Server\140\PYTHON_SERVER\lib\site-packages\revoscalepy\data\sample-data.
Pre-trained models in R and Python	Pre-trained models are supported and usable on a standalone server, but you cannot install them through SQL Server Setup. The setup program for Microsoft Machine Learning Server provides the models, which you can install free of charge. For more information, see Install pretrained machine learning models on SQL Server .

Get started step-by-step

Start with setup, attach the binaries to your favorite development tool, and write your first script.

Step 1: Install the software

Install either one of these versions:

- [SQL Server 2017 Machine Learning Server \(standalone\)](#)
- [SQL Server 2016 R Server \(Standalone\) - R only](#)

Step 2: Configure a development tool

Configure your development tools to use the Machine Learning Server binaries. For more information about Python, see [Link Python binaries](#). For instructions on how to connect in R Studio, see [Using Different Versions of R](#) and point the tool to C:\Program Files\Microsoft SQL Server\140\R_SERVER\bin\x64. You could also try [R Tools for Visual Studio](#).

Step 3: Write your first script

Write R or Python script using functions from RevoScaleR, revoscalepy, and the machine learning algorithms.

- [Explore R and RevoScaleR in 25 Functions](#): Start with basic R commands and then progress to the RevoScaleR distributable analytical functions that provide high performance and scaling to R solutions. Includes parallelizable versions of many of the most popular R modeling packages, such as k-means clustering, decision trees, and decision forests, and tools for data manipulation.
- [Quickstart: An example of binary classification with the microsoftml Python package](#): Create a binary classification model using the functions from microsoftml and the well-known breast cancer dataset.

Choose the best language for the task. R is best for statistical computations that are difficult to implement using SQL. For set-based operations over data, leverage the power of SQL Server to achieve maximum performance. Use the in-memory database engine for very fast computations over columns.

Step 4: Operationalize your solution

Standalone servers can use the [operationalization](#) functionality of the non-SQL-branded [Microsoft Machine Learning Server](#). You can configure a standalone server for operationalization, which gives you these benefits: deploy and host your code as web services, run diagnostics, test web service capacity.

See also

[SQL Server Machine Learning Services \(In-Database\)](#)

What's new in SQL Server Machine Learning Services

7/20/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Machine learning capabilities are added to SQL Server in each release as we continue to expand, extend, and deepen the integration between the data platform and the data science, analytics, and supervised learning you want to implement over your data.

New in SQL Server 2017

This release added Python support and industry-leading machine learning algorithms. Renamed to reflect the new scope, SQL Server 2017 marked the introduction of **SQL Server Machine Learning Services (In-Database)**, with language support for both Python and R.

This release also introduced **SQL Server Machine Learning Server (Standalone)**, fully independent of SQL Server, for R and Python workloads that you want to run on a dedicated system. With the standalone server, you can distribute and scale R or Python solutions without using SQL Server.

RELEASE	FEATURE UPDATE
CU 6	Bug fixes and package refresh, but no new feature announcements. Fixes include support for DateTime data types in SPEES query for Python and improved error messages in microsoftml when pre-trained models are missing.
CU 5	Bug fixes and package refresh, but no new feature announcements. Fixes include improvements to transform functions and variables in revoscalepy, correcting the long path-related errors in rxInstallPackages, fixing connections in a loopback for RxExec and rx_exec functions, and revisions to warning messages.
CU 4	Bug fixes and package refresh, but no new feature announcements.
CU 3	Python model serialization in revoscalepy, using the rx_serialize_model function . Native scoring plus enhancements to Realtime scoring . With in-database scoring, throughput is a million rows per second using R models. In this update, realtime scoring and native scoring offer better performance in single-row and batch scoring. Native scoring uses a T-SQL function for fast scoring that can be run on any edition of SQL Server, even on Linux. The function requires no installation of R or extra configuration. This means you can train a model elsewhere, save it in SQL Server, and then perform scoring without ever calling R. For more information on scoring methodologies, see How to perform realtime scoring or native scoring .
CU 2	Bug fixes and package refresh, but no new feature announcements.

RELEASE	FEATURE UPDATE
CU 1	<p>In revoscalepy, adds rx_create_col_info for returning schema information from a SQL Server data source, similar to rxCreateCollInfo for R.</p> <p>Enhancements to rx_exec to support parallel scenarios using the <code>RxLocalParallel</code> compute context.</p>
Initial release	<p>Python integration for in-database analytics</p> <p>The revoscalepy package is the Python-equivalent of RevoScaleR. You can create Python models for linear and logistic regressions, decision trees, boosted trees, and random forests, all parallelizable, and capable of being run in remote compute contexts. This package supports use of multiple data sources and remote compute contexts. The data scientist or developer can execute Python code on a remote SQL Server, to explore data or build models without moving data.</p> <p>The microsoftml package is the Python-equivalent of the MicrosoftML R package.</p> <p>T-SQL and Python integration through sp_execute_external_script. You can call any Python code using this stored procedure. This secure infrastructure enables enterprise-grade deployment of Python models and scripts that can be called from an application using a simple stored procedure. Additional performance gains are achieved by streaming data from SQL to Python processes and MPI ring parallelization.</p> <p>You can use the T-SQL PREDICT function to perform native scoring on a pre-trained model that has been previously saved in the required binary format.</p>
Initial release	<p>MicrosoftML (R) contains state-of-the-art machine learning algorithms and data transformation that can be scaled or run in remote compute contexts. Algorithms include customizable deep neural networks, fast decision trees and decision forests, linear regression, and logistic regression.</p>
Initial release	<p>Pre-trained models for image recognition and positive-negative sentiment analysis. Use these models to generate predictions on your own data.</p>
Initial release	<p>R package management, including the following highlights: database roles to help the DBA manage packages and assign permissions to install packages, CREATE EXTERNAL LIBRARY statement in T-SQL to help DBAs manage packages without needing to know R, and a rich set of R functions in RevoScaleR to help install, remove, or list packages owned by users.</p>
Initial release	<p>Operationalization through mrsdeploy for deploying and hosting R script as a web service. Applies to R script only (no Python equivalent). Intended for the (Standalone) server option to avoid resource competition with other SQL Server operations.</p>

New in SQL Server 2016

This release introduced machine learning capabilities into SQL Server through **SQL Server 2016 R Services**, an in-database analytics engine for processing R script on resident data within a database engine instance.

Additionally, **SQL Server 2016 R Server (Standalone)** was released as a way to install R Server on a Windows server. Initially, SQL Server Setup provided the only way to install R Server for Windows. In later releases, developers and data scientists who wanted R Server on Windows could use another standalone installer to achieve the same goal. The standalone server in SQL Server is functionally equivalent to the standalone server product, [Microsoft R Server for Windows](#).

RELEASE	FEATURE UPDATE
CU additions	<p>Realtime scoring relies on native C++ libraries to read a model stored in an optimized binary format, and then generate predictions without having to call the R runtime. This makes scoring operations much faster. With realtime scoring, you can run a stored procedure or perform realtime scoring from R code. Realtime scoring is also available for SQL Server 2016, if the instance is upgraded to the latest release of Microsoft R Server.</p>
Initial release	<p>R integration for in-database analytics.</p> <p>R packages for calling R functions in T-SQL, and vice versa. RevoScaleR functions provide R analytics at scale by chunking data into component parts, coordinating and managing distributed processing, and aggregating results. In SQL Server 2016 R Services (In-Database), the RevoScaleR engine is integrated with a database engine instance, bringing data and analytics together in the same processing context.</p> <p>T-SQL and R integration through sp_execute_external_script. You can call any R code using this stored procedure. This secure infrastructure enables enterprise-grade deployment of Rn models and scripts that can be called from an application using a simple stored procedure. Additional performance gains are achieved by streaming data from SQL to R processes and MPI ring parallelization.</p> <p>You can use the T-SQL PREDICT function to perform native scoring on a pre-trained model that has been previously saved in the required binary format.</p>

Linux support roadmap

Machine learning using R or Python in-database is not currently supported in SQL Server on Linux. Look for announcements in a later release.

However, on Linux you can perform [native scoring](#) using the T-SQL PREDICT function. Native scoring lets you score from a pretrained model very fast, without calling or even requiring an R runtime. This means you can use SQL Server on Linux to generate predictions very fast, to serve client applications.

Azure SQL Database roadmap

There is limited support for R in Azure SQL Database: available only in West Central US, in services created at the Premium tier. Expanded coverage, including Python support, is likely to follow in a future release. However, there is no projected release date at this time.

Next steps

- [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#)
- [Machine learning tutorials and samples](#)

Feature availability across editions of SQL Server Machine Learning Services

4/12/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Machine learning features are available in SQL Server 2016 and SQL Server 2017. This article lists the editions providing the feature, describes limitations that apply in specific editions, and lists capabilities available only in certain editions.

SQL Server 2017 features

Enterprise and Developer editions have the same feature coverage so that you can build solutions for an Enterprise installation without incurring the same cost. Although the editions are functionally equivalent, use of the Developer Edition is not supported for production environments.

The difference between basic and advanced integration is scale. Advanced integration can use all available cores for parallel processing of data sets at any size your computer can accommodate. Basic integration is limited to 2 cores and to data sets fitting in memory.

Basic and advanced integration applies to the (In-Database) instances. A standalone server is not a database engine instance feature and is offered as an installation option only in Developer and Enterprise editions.

FEATURE	ENTERPRISE	STANDARD	WEB	EXPRESS WITH ADVANCED SERVICES	EXPRESS
Basic R integration	Yes	Yes	Yes	Yes	No
Advanced R integration	Yes	No	No	No	No
Basic Python integration	Yes	Yes	Yes	Yes	No
Advanced Python integration	Yes	No	No	No	No
Machine Learning Server (Standalone)	Yes	No	No	No	No

NOTE

Only a (Standalone) server offers the [operationalization](#) features that are included in a Microsoft (non-SQL-branded) R Server or Machine Learning Server installation. Operationalization includes web service deployment and hosting capabilities.

For an (In-Database) installation, the equivalent approach to operationalizing solutions is leveraging the capabilities of the database engine, when you convert code to a function that can run in a stored procedure.

SQL Server 2016 R features

SQL Server 2016 includes R integration only. In SQL Server 2016, basic and advanced R integration are equivalent to SQL Server 2017.

R feature availability in Azure SQL Database

After an initial test release, R Services was removed from Azure SQL Database, pending further development.

Performance expectations for Enterprise Edition

Performance of machine learning solutions in SQL Server is expected to generally surpass implementations using conventional R, given the same hardware. That is because, in SQL Server, R solutions can be run using server resources and sometimes distributed to multiple processes using the **RevoScaleR** functions.

Users can also expect to see considerable differences in performance and scalability for the same machine learning solution if run in Enterprise Edition vs. Standard Edition. Reasons include support for parallel processing, increased threads available for machine learning, and streaming (or chunking), which allows the RevoScaleR functions to handle more data than can fit in memory.

However, performance even on identical hardware can be affected by many factors outside the R or Python code. These factors include competing demands on server resources, the type of query plan that is created, schema changes, the need to update statistics or create a new query plan, fragmentation, and more. It is possible that a stored procedure containing R or Python code might run in seconds under one workload, but take minutes when there are other services running. Therefore, we recommend that you monitor multiple aspects of server performance, including networking for remote compute contexts, when measuring machine learning performance.

We also recommend that you configure [Resource Governor](#) (available in Enterprise Edition) to customize the way that external script jobs are prioritized or handled under heavy server workloads. You can define classifier functions to specify the source of the external script job and prioritize certain workloads, limit the amount of memory used by SQL queries, and control the number of parallel processes used on a workload basis.

See also

- [Editions and components of SQL Server 2016](#)
- [Editions and components of SQL Server 2017](#)
- [Tips on computing with big data in R \(Machine Learning Server\)](#)

Architecture overview for SQL Server Machine Learning Services

4/11/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✔ SQL Server (Windows only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

This article describes the goals of the extensibility framework that supports execution of Python and R script in SQL Server.

It also provides an overview of how the architecture is designed to meet these goals, how R and Python are supported and executed by SQL Server, and the benefits of integration.

Overall, the extensibility framework is almost identical for R and Python, with some minor differences in the details of the launchers that are called, configuration options, and so forth. For further information about the implementation for a specific language, see these articles:

- [Architecture Overview for SQL Server R Services](#)
- [Architecture Overview for Python in SQL Server](#)

Background

In SQL Server 2016, numerous changes were introduced to the database engine to support execution of R scripts using SQL Server. In SQL Server 2017, this underlying infrastructure was improved to add support for the Python language.

The goal of the extensibility framework was to create a better interface between SQL Server and data science languages such as R and Python, both to reduce the friction that occurs when data science solutions are moved into production, and to protect data that might be exposed during the data science development process.

By executing a trusted scripting language within a secure framework managed by SQL Server, the database developer can maintain security while allowing data scientists to use enterprise data.



- Current users of R or Python should be able to port their code and execute it in SQL Server with relatively minor modifications.
- The data security model in SQL Server is extended to data used by external script languages. In other words, someone running R or Python script should not be able to use any data that could not be accessed by that user in a SQL query.
- The database administrator should be able to manage resources used by external scripts, manage users, and manage and monitor external code libraries.
- The system must support solutions based entirely on open source distributions of R and Python, but use proprietary components developed by Microsoft to provide greater security and performance.

Architecture core concepts

To meet these goals, the architecture of SQL Server 2016 R Services and SQL Server 2017 Machine Learning Services for R and Python is based on these core concepts:

- **Multi-process architecture**

Both R and Python are open-source languages with rich and enthusiastic community support. Therefore, it is important to maintain full interoperability with open source R and Python.

Open source distributions of R and Python are installed with SQL Server under license, and can function independently from SQL Server if needed.

In addition, Microsoft provides a set of proprietary libraries that provide integration with SQL Server, including data translation, compression, and optimization targeted at each supported language.

- **Security**

Better security means support for both integrated Windows authentication and password-based SQL logins, as well as secure handling of credentials, reliance on SQL Server for data protection, and use of the SQL Server Trusted Launchpad to manage external script execution and secure data used in scripts.

- **Scalability and performance**

Integration with SQL Server is key to improving the usefulness of R and Python in the enterprise. Any R or Python script can be run by calling a stored procedure, and the results are returned as tabular results directly to SQL Server, making it easy to generate or consume machine learning from any application that can send a SQL query and handle the results.

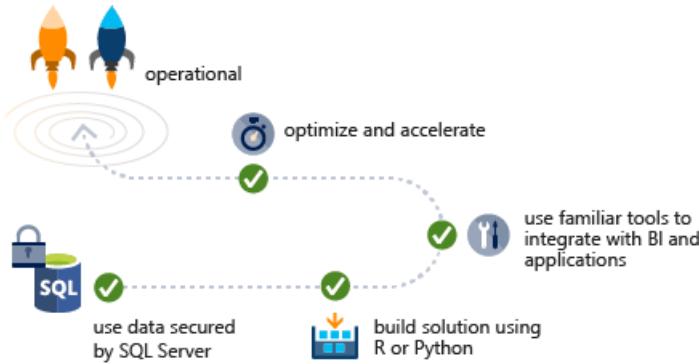
Performance optimization relies on two equally powerful aspects of the platform: resource governance and parallel processing using SQL Server, and distributed computing provided by the algorithms in **RevoScaleR** and **revoscalepy**.

Solution development and deployment

In addition to these core goals for the extensibility platform, the machine learning services in SQL Server are designed to provide strong integration with the database engine and the BI stack, with these benefits:

- Performance and resource management through traditional monitoring tools
- Easy use of Python and R data by BI suites or any application that can consume SQL query results
- Much lower barrier for enterprise development of machine learning solutions

Let's see how it works in practice.



1. Data is kept within the compliance boundary and use of data can be managed and monitored by SQL Server. Meanwhile, the DBA has full control over who can install packages or run scripts on the server. If so desired, the DBA can also delegate permissions on a database level to data scientists or managers.

2. Data scientists can build and test solutions in their preferred R or Python environments, disconnected from the server.
3. The SQL developer can use familiar tools such as Management Studio or Visual Studio to integrate the R or Python code with SQL Server. The tight integration means that the savvy developer can choose the best tool to optimize each task. For example, you might use SQL for some feature engineering tasks and R for others. You might embed Python script in an Integration Services task to perform sophisticated text analytics.
4. Tested and ready-to-deploy solutions can be optimized using SQL Server technologies, such as columnstore indexes, for better performance. Newer features let you batch-train many small models in parallel on partitioned data set, or score millions of rows in using native SQL code optimized for machine learning tasks.
5. Ready to lift off? You can easily expose your predictive solutions to the BI stack or external applications by using stored procedures.

Related products

Not sure which machine learning solution meets your needs? In addition to embedded analytics in SQL Server 2016 and SQL Server 2017, Microsoft provides the following machine learning platforms and services:

- [Microsoft R Server and Machine Learning Server](#)

A multi-platform environment for developing, distributing, and managing machine learning jobs

- [Data Science Virtual Machine](#)

All the tools you need for machine learning, preinstalled. Use Jupyter notebooks, Python, or R.

Try the new [Windows 2016 preview edition](#), which includes GPU versions of popular deep learning frameworks such as CNTK and mxNet, as well as support for Windows containers!

- [Azure Cognitive Services](#)

A variety of cloud services for adding AI and ML into your applications, including natural language indexing of video, facial recognition, emotion detection, text analytics, machine translation, and much, more

- [Azure Machine Learning](#)

A cloud-based drag-and-drop interface for designing machine learning workflows, coupled with the ability to automate and integrate with applications via web services and PowerShell

See Also

[Compare Machine Learning Server and Microsoft R products](#)

Architecture overview for R in SQL Server

4/12/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This section provides an overview of the architecture of SQL Server 2016 R Services, and of SQL Server 2017 Machine Learning Services.

The architecture for the extensibility architecture is the same or very similar for the SQL Server 2016 and SQL Server 2017 releases, and similar also for R and Python. However, to simplify the discussion, this article discusses only the R components, including new components added in the SQL Server database engine to support external script execution, security, R libraries, and interoperability with open source R.

Additional details are provided in the links for each section.

R interoperability

Both SQL Server 2016 R Services and SQL Server 2017 Machine Learning Services (In-Database) install an open source distribution of R, as well as packages provided by Microsoft that support distributed and/or parallel processing.

The architecture is designed such that external scripts using R run in a separate process from SQL Server. Current users of R should be able to port their R code and execute it in T-SQL with relatively minor modifications.

To scale your solution or use parallel processing, we recommend that you use the [RevoScaleR](#) package or the [MicrosoftML](#) package. If you do not use the distributed computing capabilities provided by these libraries, you can still get some performance improvements by running your R code in the context of SQL Server.

For more information about the external scripting components that are installed, or the interaction of SQL Server with R, see [R Interoperability](#)

Components to support R integration

The extensibility framework introduced in SQL Server 2016 is continued in SQL Server 2017. The extensibility components are used by SQL Server to start the external runtime for R, to pass data between R and the database engine, and to coordinate parallel tasks needed for an R job.

The role of these additional components is to improve data exchange speed and compression, while providing a secure, high-performance platform for running external scripts.

For detailed description of the components that support R, such as the SQL Server Trusted Launchpad and RLauncher, see [New Components](#).

Security

When you run R code using Machine Learning Services or SQL Server R Services, all R scripts are executed outside the SQL Server process, to provide security and greater manageability. This isolation of processes holds true regardless of whether you run the R script as part of a stored procedure, or connect to the SQL Server computer job from a remote computer and start a job that uses the server as the compute context.

SQL Server intercepts all job requests, secures the task and its data using Windows job objects, and maintains security over data using SQL Server user accounts and database roles.

Data is kept within the compliance boundary by enforcing SQL Server security at the table, database, and instance level. The database administrator can control who has the ability to run R jobs, and who has the ability to install or share R packages. The administrator can also monitor the use of R scripts by either remote or local users, and monitor and manage the resources consumed.

Next steps

[Components that support R integration](#)

[R interoperability](#)

[Security overview](#)

R interoperability in SQL Server

7/11/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic focuses on the mechanism for running R within SQL Server, and describes the differences between Microsoft R and open source R.

Applies to: SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

For information about additional components, see [New Components in SQL Server](#).

Open source R components

R Services (In-Database) includes a complete distribution of the base R packages and tools. For more information about what is included with the base distribution, see the documentation installed during setup in the following default location: `C:\Program Files\Microsoft SQL Server\<instance_name>\R_SERVICES\doc\manual`

As part of the installation of R Services (In-Database), you must consent to the terms of the GNU Public License. Thereafter, you can run standard R packages without further modification just as you would in any other open source distribution of R.

SQL Server does not modify the R runtime in any way. The R runtime is executed outside of the SQL Server process and can be run independently of SQL Server. However, we strongly recommend that you do not run these tools while SQL Server is using R, to avoid resource contention.

The R base package distribution that is associated with a specific SQL Server instance can be found in the folder associated with the instance. For example, if you installed R Services on the default instance, the R libraries are located in this folder by default:

```
C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library
```

Similarly, the R tools associated with the default instance would be located in thi folder by default:

```
C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\bin
```

For more information about how Microsoft R is different from a base distribution of R that you might get from CRAN, see [Interoperability with R language and Microsoft R products and features](#)

Additional R packages from Microsoft R

In addition to the base R distribution, R Services (In-Database) includes some proprietary R packages as well as a framework for parallel execution of R that also supports execution of R in remote compute contexts.

This combined set of R features - the R base distribution plus the enhanced R features and packages - is referred to as **Microsoft R**. If you install Microsoft R Server (Standalone), you get exactly the same set of packages that are installed with SQL Server R Services (In-Database) but in a different folder.

Microsoft R includes a distribution of the Intel Math Kernel Library, which is used whenever possible for faster mathematical processing. For example, the basic linear algebra (BLAS) library is used for many of the add-on packages as well as for R itself. For more information, see these articles:

- [How the Intel Math Kernel Speeds up R](#)

- [Performance benefits of linking R to multithreaded math libraries](#)

Among the most important additions to Microsoft R are the **RevoScaleR** and **RevoPemaR** packages. These are R packages that have been written largely in C or C++ for better performance.

- **RevoScaleR.** Includes a variety of APIs for data manipulation and analysis. The APIs have been optimized to analyze data sets that are too big to fit in memory and to perform computations distributed over several cores or processors.

The RevoScaleR APIs also support working with subsets of data for greater scalability. In other words, most RevoScaleR functions can operate on chunks of data, and use updating algorithms to aggregate results. Therefore R solutions based on the RevoScaleR functions can work with very large data sets and are not bound by local memory.

The RevoScaleR package also supports the .XDF file format for faster movement and storage of data used for analysis. The XDF format uses columnar storage, is portable, and can be used to load and then manipulate data from various sources, including text, SPSS, or an ODBC connection.

- **RevoPemaR.** PEMA stands for Parallel External Memory Algorithm. The **RevoPemaR** package provides APIs that you can use to develop your own parallel algorithms. For more information, see [RevoPemaR Getting Started Guide](#).

We also recommend that you try [MicrosoftML](#), a new package from Microsoft R that supports remote execution of R code and scalable, distributed processing, using improved machine learning algorithms developed by Microsoft Research.

Next steps

[Architecture overview](#)

[Components in SQL Server to support R](#)

[Security overview](#)

Components in SQL Server to support R

4/12/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In SQL Server 2016 and 2017, the database engine includes optional components that support extensibility for external script languages, including R and Python. Support for the R language was added in SQL Server 2016; support for Python was added in SQL Server 2017 Machine Learning Services.

This topic describes the new components that work specifically with the R language. For a discussion of how these components work with open source R, see [R Interoperability](#)

Components and providers

In addition to the shell that loads R and executes R code as described in the architecture overview, SQL Server includes these additional components.

Launchpad

The SQL Server Trusted Launchpad is a service provided by SQL Server 2017 for supporting execution of external scripts, similar to the way that the full-text indexing and query service launches a separate host for processing full-text queries.

The Launchpad service will start only trusted launchers that are published by Microsoft, or that have been certified by Microsoft as meeting requirements for performance and resource management. The naming for the language-specific launchers is straightforward:

- R - RLauncher.dll
- Python - PythonLauncher.dll

The SQL Server Trusted Launchpad service runs under its own user account. Each satellite process for a specific language runtime will inherit the user account of the Launchpad. For more information about the configuration and security context of the Launchpad, see [Security overview](#).

BxlServer and SQL Satellite

BxlServer is an executable provided by Microsoft that manages communication between SQL Server and the R runtime and also implements RevoScaleR functions. It creates the Windows job objects that are used to contain R sessions, provisions secure working folders for each R job, and uses SQL Satellite to manage data transfer between R and SQL Server.

In effect, BxlServer is a companion to R that works with SQL Server to support integration of R with SQL Server. BXL stands for Binary Exchange language and refers to the data format used to move data efficiently between SQL Server and external processes such as R. BxlServer.dll is also installed when you install Microsoft R Client or Microsoft R Server.

The SQL Satellite is a new extensibility API in SQL Server 2016 that is provided by the database engine to support external code or external runtimes implemented using C or C++. BxlServer uses SQL Satellite for communicating with SQL Server.

The SQL Satellite uses a custom data format that is optimized for fast data transfer between SQL Server and external script languages. It performs type conversions and defines the schemas of the input and output datasets during communications between SQL Server and R.

BxlServer uses SQL Satellite for these tasks:

- Reading input data
- Writing output data
- Getting input arguments
- Writing output arguments
- Error handling
- Writing standard output and errors back to the client

The SQL Satellite can be monitored by using Extended Events. For more information, see [Extended Events for SQL Server R Services](#).

Communication channels between components

- **TCP/IP**

By default, internal communications between SQL Server and the SQL Satellite use TCP/IP.

- **Named Pipes**

Internal data transport between the BxlServer and SQL Server via SQL Satellite uses a proprietary, compressed data format to enhance performance. Data from R memory to BxlServer is exchanged over named pipes in BXL format.

- **ODBC**

Communications between external data science clients and the SQL Server instance use ODBC. The account that sends the R jobs to SQL Server must have both permissions to connect to the instance and to run external scripts. Additionally, the account must have permission to access any data used by the job, to write data (for example, if saving results to a table), or to create database objects (for example, if saving R functions as part of a new stored procedure).

When SQL Server is used as the compute context for an R job sent from a remote client, and the R command must retrieve data from an external source, ODBC is used for writeback. SQL Server will map the identity of the user issuing the remote R command to the identity of the user on the current instance, and run the ODBC command using that user's credentials. The connection string needed to perform this ODBC call is obtained from the client code.

Additional ODBC calls can be made inside the script by using **RODBC**. RODBC is a popular R package used to access data in relational databases; however, its performance is generally slower than comparable providers used by SQL Server. Many R scripts use embedded calls to RODBC as a way of retrieving "secondary" datasets for use in analysis. For example, the stored procedure that trains a model might define a SQL query to get the data for training a model, but use an embedded RODBC call to get additional factors, to perform lookups, or to get new data from external sources such as text files or Excel.

The following code illustrates an RODBC call embedded in an R script:

```
library(RODBC);
connStr <- paste("Driver=SQL Server;Server=", instance_name, ";Database=", database_name,
";Trusted_Connection=true;", sep="");
dbhandle <- odbcDriverConnect(connStr)
OutputDataSet <- sqlQuery(dbhandle, "select * from table_name");
```

- **Other protocols**

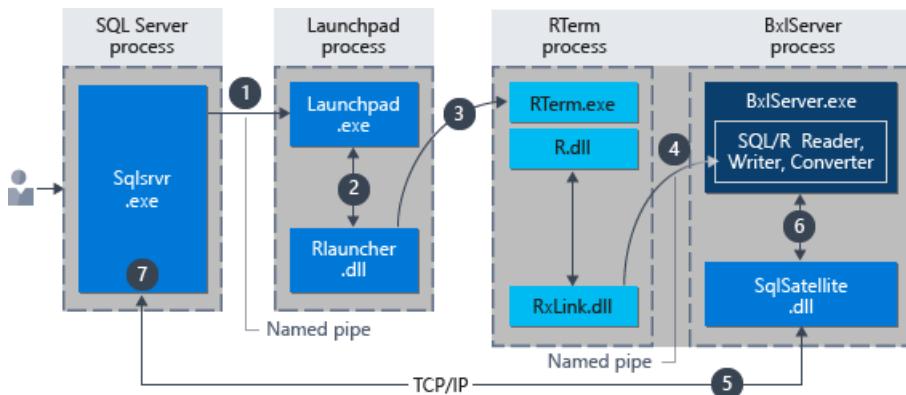
Processes that might need to work in "chunks" or transfer data back to a remote client can also use the .XDF format supported by Microsoft R. Actual data transfer is via encoded blobs.

Interaction of components

The component architecture just described has been provided to guarantee that open source R code can work "as is", while providing greatly increased performance for code that runs on a SQL Server computer. The mechanism of how the components interact with the R runtime and the SQL Server database engine differs somewhat, depending on how the R code is launched. The key scenarios are summarized in this section.

R scripts executed from SQL Server in-database

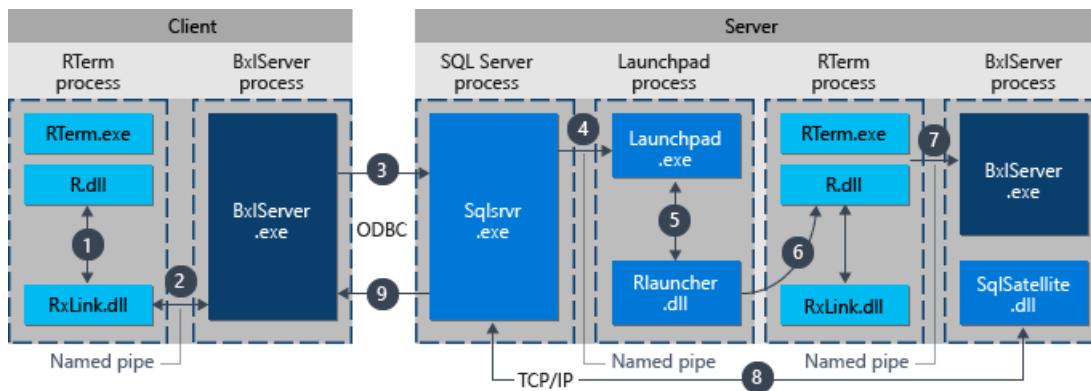
R code that is run from "inside" SQL Server is executed by calling a stored procedure. Thus, any application that can make a stored procedure call can initiate execution of R code. Thereafter SQL Server manages the execution of R code as summarized in the following diagram.



1. A request for the R runtime is indicated by the parameter `@language='R'` passed to the stored procedure, `sp_execute_external_script`. SQL Server sends this request to the Launchpad service.
2. The Launchpad service starts the appropriate launcher; in this case, RLauncher.
3. RLauncher starts the external R process.
4. BxlServer coordinates with the R runtime to manage exchanges of data with SQL Server and storage of working results.
5. SQL Satellite manages communications about related tasks and processes with SQL Server.
6. BxlServer uses SQL Satellite to communicate status and results to SQL Server.
7. SQL Server gets results and closes related tasks and processes.

R scripts executed from a remote client

When connecting from a remote data science client that supports Microsoft R, you can run R functions in the context of SQL Server by using the RevoScaleR functions. This is a different workflow from the previous one, and is summarized in the following diagram.



1. For RevoScaleR functions, the R runtime calls a linking function which in turn calls BxlServer.
2. BxlServer is provided with Microsoft R and runs in a separate process from the R runtime.
3. BxlServer determines the connection target and initiates a connection using ODBC, passing credentials supplied as part of the connection string in the R data source object.

4. BxlServer opens a connection to the SQL Server instance.
5. For an R call, the Launchpad service is invoked, which in turn starts the appropriate launcher, RLauncher. Thereafter, processing of R code is similar to the process for running R code from T-SQL.
6. RLauncher makes a call to the instance of the R runtime that is installed on the SQL Server computer.
7. Results are returned to BxlServer.
8. SQL Satellite manages communication with SQL Server and cleanup of related job objects.
9. SQL Server passes results back to the client.

Next steps

[Architecture overview](#)

[Security overview](#)

[Security considerations](#)

Security for SQL Server machine learning and R

4/12/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes the overall security architecture that is used to connect the SQL Server database engine and related components to the R runtime. Examples of the security process are provided for these common scenarios for using R in an enterprise environment:

- Executing RevoScaleR functions in SQL Server from a data science client
- Running R directly from SQL Server using stored procedures

Security overview

A SQL Server login or Windows user account is required to run R scripts that use SQL Server data or that run with SQL Server as the compute context. This requirement applies to both R Services (In-Database) and SQL Server 2017 Machine Learning Services (In-Database).

The login or user account identifies the *security principal*, who might need multiple levels of access, depending on the R script requirements:

- Permission to access the database where R is enabled
- Permissions to read data from secured objects such as tables
- The ability to write new data to a table, such as a model, or scoring results
- The ability to create new objects, such as tables, stored procedures that use R script, or custom functions that use R job
- The right to install new packages on the SQL Server computer, or use R packages provided to a group of users.

Therefore, each person who runs R code using SQL Server as the execution context must be mapped to a login in the database. Under SQL Server security, it is generally easiest to create roles to manage sets of permissions, and assign users to those roles, rather than individually set user permissions.

As an example, assume that you created some R code that runs on your laptop, and you want to run that code on SQL Server. You can do this only if these conditions are met:

- The database allows remote connections.
- A SQL login with the name and the password that you used in the R code has been added to the SQL Server at the instance level. Or, if you are using Windows integrated authentication, the Windows user specified in the connection string must be added as a user on the instance.
- The SQL login or Windows user must have the permission to execute external scripts. Generally, this permission can only be added by a database administrator.
- The SQL login or Window user must be added as a user, with appropriate permissions, in each database where the R job performs any of these operations:
 - Retrieving data
 - Writing or updating data
 - Creating new objects, such as tables or stored procedures

After the login or Windows user account has been provisioned and given the necessary permissions, you can run R code on SQL Server by using an R data source object or by calling a stored procedure. Whenever R is started

from SQL Server, the database engine security gets the security context of the user who started the R job or ran the stored procedure, and manages the mappings of the user or login to securable objects.

Hence, all R jobs that are initiated from a remote client must specify the login or user information as part of the connection string.

Interaction of SQL Server security and Launchpad security

When an R script is executed in the context of the SQL Server computer, the SQL Server Trusted Launchpad service gets an available worker account (a local user account) from a pool of worker accounts established for external process and uses that worker account to perform the related tasks.

For example, if you start an R job under your Windows domain credentials, your account might be mapped to the Launchpad worker account *SQLRUser01*.

After mapping to a worker account, SQL Server Trusted Launchpad creates a user token that is used to start processes.

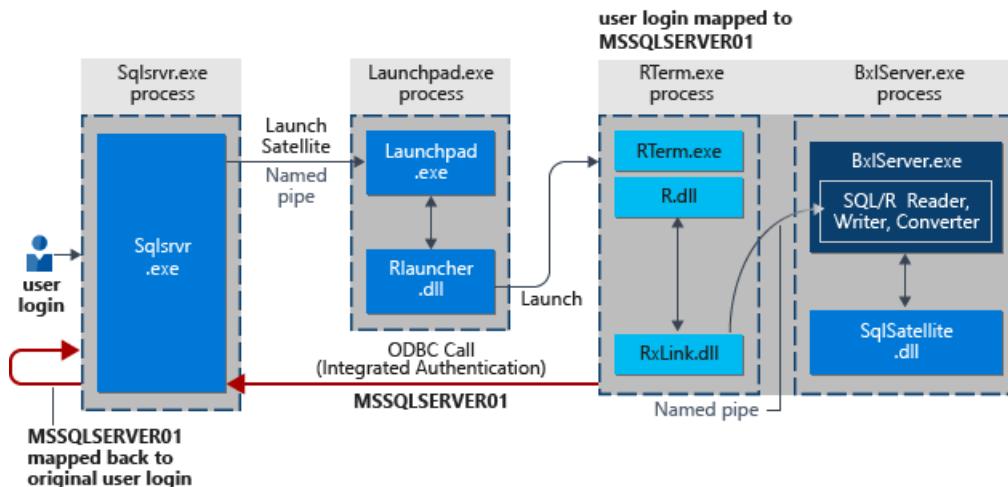
When all SQL Server operations are completed, the user worker account is marked as free and returned to the pool.

For more information about SQL Server Trusted Launchpad, see [Components in SQL Server to support R integration](#).

Implied authentication

Implied authentication is the term used for the process under which SQL Server gets the user credentials and then executes all external script tasks on behalf of the users, assuming the user has the correct permissions in the database. Implied authentication is particularly important if the R script needs to make an ODBC call outside the SQL Server database. For example, the code might retrieve a shorter list of factors from a spreadsheet or other source.

For such loopback calls to succeed, the group that contains the worker accounts, *SQLRUserGroup*, must have "Allow Log on locally" permissions. By default, this right is given to all new local users, but in some organizations stricter group policies might be enforced.



Security of worker accounts

The mapping of an external Windows user or valid SQL login to a worker account is valid only for the lifetime of the lifetime of the SQL query that runs the R script.

Parallel queries from the same login are mapped to the same user worker account.

The directories used for the processes are managed by the SQL Server Trusted Launchpad using *RLauncher*, and

directories are access-restricted. The worker account cannot access any files in folders above its own, but it can read, write, or delete children under the session working folder that was created for the SQL query with the R script.

For more information about how to change the number of worker accounts, account names, or account passwords, see [Modify the user account pool for SQL Server machine learning](#).

Security isolation for multiple external scripts

The isolation mechanism is based on physical user accounts. As satellite processes are started for a specific language runtime, each satellite task uses the worker account specified by the SQL Server Trusted Launchpad. If a task requires multiple satellites, for example, in the case of parallel queries, a single worker account is used for all related tasks.

No worker account can see or manipulate files used by other worker accounts.

If you are an administrator on the computer, you can view the directories created for each process. Each directory is identified by its session GUID.

See also

[Architecture overview for SQL Server machine learning](#)

Architecture overview for Machine Learning Services with Python

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides an overview of how Python is integrated with SQL Server, including the security model, the components in the database engine that support external script execution, and new components that enable interoperability of Python with SQL Server. For details, see the linked articles.

IMPORTANT

Support for Python is available beginning with SQL Server 2017 CTP 2.0. This pre-release feature is subject to change.

Python interoperability

SQL Server Machine Learning Services (In-Database) installs the Anaconda distribution of Python, and the Python 3.5 runtime and interpreter. This ensures near-complete compatibility with standard Python solutions. Python runs in a separate process from SQL Server, to guarantee that database operations are not compromised.

For more information about the interaction of SQL Server with Python, see [Python interoperability](#)

Components that support Python integration

The extensibility framework introduced in SQL Server 2016 now supports execution of Python script, through the addition of new language-specific components. These components improve data exchange speed and compression, while providing a secure, high-performance platform for running external scripts.

For detailed description of the components that support Python, such as the SQL Server Trusted Launchpad and PythonLauncher, see [New Components](#).

Security

Python tasks execute outside the SQL Server process, to provide security and greater manageability.

All tasks are secured by Windows job objects or SQL Server security. Data is kept within the compliance boundary by enforcing SQL Server security at the table, database, and instance level. The database administrator can control who has the ability to run Python jobs, monitor the use of Python scripts by users, and view the resources consumed.

For details, see [Security for Python](#)

Resource governance

In SQL Server Enterprise Edition, you can use Resource Governor to manage and monitor resource use of external script operations, including R script and Python scripts.

For more information, see [Resource Governance for R](#).

Next steps

[Run Python using T-SQL](#)

SQL Server Machine Learning Services with Python

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Python is a language that offers great flexibility and power for a variety of machine learning tasks. Open source libraries for Python include several platforms for customizable neural networks, as well as popular libraries for natural language processing. Now, this widely-used language is supported in SQL Server 2017 Machine Learning.

Because Python is integrated with the SQL Server database engine, you can keep analytics close to the data and eliminate the costs and security risks associated with data movement. You can deploy machine learning solutions based on Python using convenient, familiar tools such as Visual Studio. Your production applications can get predictions, models, or visuals from the Python 3.5 runtime by simply calling a T-SQL stored procedure.

This release includes the Anaconda distribution of Python, as well as the [revoscalepy](#) library, to improve the scale and performance of your machine learning solutions.

You can install everything you need to get started with Python through SQL Server 2017 setup:

- **Machine Learning Services (In-Database):** Install this feature, together with the SQL Server database engine, to enable secure execution of Python scripts on the SQL Server computer.

When you select this feature, extensions are installed in the database engine to support execution of Python scripts, and a new service is created, the SQL Server Trusted Launchpad, to manage communications between the Python runtime and the SQL Server instance.

- **Machine Learning Server (Standalone):** If you do not need SQL Server integration, install this feature to get Python and R support for distributed machine learning.

See also

- [SQL Server Machine Learning and R Services \(In-Database\)](#)
- [SQL Server Machine Learning and R Server \(Standalone\)](#)
- [Python architecture](#)
- [Python Tutorials](#)

Python interoperability with SQL Server

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes the Python components that are installed if you enable the feature **Machine Learning Services (In-Database)** and select Python as the language.

Python components

SQL Server does not modify the Python executables. The Python runtime is installed independently of SQL tools, and is executed outside of the SQL Server process.

The distribution that is associated with a specific SQL Server instance can be found in the folder associated with the instance.

For example, if you installed Machine Learning Services with the Python option on the default instance, look under:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES
```

Installation of SQL Server 2017 Machine Learning Services adds the Anaconda distribution of Python. Specifically, the Anaconda 3 installers are used, based on the Anaconda 4.3 branch. The expected Python level for SQL Server 2017 is Python 3.5.

New Python packages in this release

For a list of packages supported by the Anaconda distribution, see the Continuum analytics site: [Anaconda package list](#)

Machine Learning Services in SQL Server 2017 also includes the new **revoscalepy** library for Python.

This library provides functionality equivalent to that of the **RevoScaleR** package for Microsoft R. In other words, it supports creation of remote compute contexts, as well as a various scalable machine learning models, such as **rxLinMod**. For more information about RevoScaleR, see [Distributed and parallel computing with ScaleR](#).

The **microsoftml** for Python package is installed as part of SQL Server Machine Learning when you add Python to your installation. This package contains many machine learning algorithms that have been optimized for speed and accuracy, as well as in-line transformations for working with text and images. for details, see [Using the MicrosoftML package with SQL Server](#).

Microsoftml and revoscalepy are tightly coupled; data sources used in microsoftml are defined as revoscalepy objects. Compute context limitations in revoscalepy transfer to microsoftml. Namely, all functionality is available for local operations, but switching to a remote compute context requires RxInSqlServer.

Using Python in SQL Server

You import the **revoscalepy** module into your Python code, and then call functions from the module, like any other Python functions.

Input data for Python must be tabular. All Python results must be returned in the form of a **pandas** data frame.

You can execute your Python code inside T-SQL, by embedding the script in a stored procedure.

Or, run the code from a local Python IDE and have the script executed on the SQL Server computer, by defining a remote compute context.

You can work with local data, get data from SQL Server or other ODBC data sources, or use the XDF file format to exchange data with other sources, or with R solutions.

For more information

- Supported functions: [What is revoscalepy](#)
- Supported Python data types: [Python Libraries and Data Types](#)
- Supported data sources: ODBC databases, SQL Server, and XDF files
- Supported compute contexts: local, or SQL Server

Licensing

As part of the installation of Machine Learning Services with Python, you must consent to the terms of the GNU Public License.

See Also

[Python libraries and data types](#)

Components in SQL Server to support Python integration

4/11/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Beginning in SQL Server 2017, Machine Learning Services supports Python as an external language that can be executed from T-SQL, or executed remotely using SQL Server as the compute context.

This topic describes the components in SQL Server 2017 that support extensibility in general and the Python language specifically.

SQL Server components and providers

To configure SQL Server 2017 to allow Python script execution is a multi-step process.

1. Install the extensibility feature.
2. Enable the external script execution feature.
3. Restart the database engine service.

Additional steps might be required to support remote script execution.

For more information, see [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#).

Launchpad

The SQL Server Trusted Launchpad is a service introduced in SQL Server 2016 that manages and executes external scripts, similar to the way that the full-text indexing and query service launches a separate host for processing full-text queries.

The Launchpad service can start only trusted launchers that are published by Microsoft, or that have been certified by Microsoft as meeting requirements for performance and resource management.

- SQL Server 2017 supports R and Python 3.5
- SQL Server 2016 supports R

The SQL Server Trusted Launchpad service runs under its own user account.

TIP

If you change the account that runs Launchpad, be sure to do so using SQL Server Configuration Manager, to ensure that changes are written to related files.

To execute tasks in a specific supported language, the Launchpad gets a secured worker account from the pool, and starts a satellite process to manage the external runtime:

- RLauncher.dll for the R language
- Pythonlauncher.dll for Python 3.5

Each satellite process inherits the user account of the Launchpad and uses that worker account for the duration of script execution. If the Python script uses parallel processes, they are created under the same, single worker account.

For more information about the security context of the Launchpad, see [Security](#).

BxlServer and SQL Satellite

If you run [Process Explorer](#) while a Python job is running, you might see one or multiple instances of BxlServer.

BxlServer is an executable provided by Microsoft that manages communication between SQL Server and Python (or R). It creates the Windows job objects that are used to contain external script sessions, provisions secure working folders for each external script job, and uses SQL Satellite to manage data transfer between the external runtime and SQL Server.

In effect, BxlServer is a companion to Python that works with SQL Server to transfer data and manage tasks. BXL stands for Binary Exchange language and refers to the data format used to move data efficiently between SQL Server and external processes. BxlServer is also an important part of Microsoft R Client and Microsoft R Server.

SQL Satellite is an extensibility API, included in the database engine starting with SQL Server 2016, that supports external code or external runtimes implemented using C or C++.

BxlServer uses SQL Satellite for these tasks:

- Reading input data
- Writing output data
- Getting input arguments
- Writing output arguments
- Error handling
- Writing STDOUT and STDERR back to client

SQL Satellite uses a custom data format that is optimized for fast data transfer between SQL Server and external script languages. It performs type conversions and defines the schemas of the input and output datasets during communications between SQL Server and the external script runtime.

The SQL Satellite can be monitored by using windows extended events (xEvents). For more information, see [Extended Events for R](#).

Communication channels between components

• TCP/IP

By default, internal communications between SQL Server and the SQL Satellite use TCP/IP.

• Named Pipes

Internal data transport between the BxlServer and SQL Server through SQL Satellite uses a proprietary, compressed data format to enhance performance. Data is exchanged between Python and BxlServer in BXL format, using Named Pipes.

• ODBC

Communications between external data science clients and the SQL Server instance use ODBC. The account that sends the script jobs to SQL Server must have both permissions to connect to the instance and to run external scripts.

Additionally, depending on the task, the account might need these permissions:

- Read data used by the job
- Write data to tables: for example, when saving results to a table
- Create database objects: for example, if saving external script as part of a new stored procedure.

When SQL Server is used as the compute context for Python script executed from a remote client, and the

Python executable must retrieve data from an external source, ODBC is used for writeback. SQL Server maps the identity of the user issuing the remote command to the identity of the user on the current instance, and runs the ODBC command using that user's credentials. The connection string needed to perform this ODBC call is obtained from the client code.

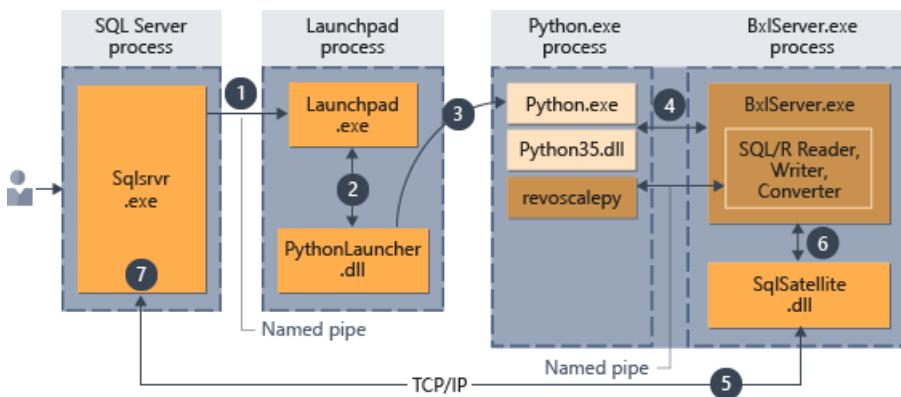
Interaction of components

The following diagrams depict the interaction of SQL Server components with the Python runtime in each of the supported scenarios: running script in-database, and remote execution from a Python terminal, using a SQL Server compute context.

Python scripts executed in-database

When you run Python "inside" SQL Server, you must encapsulate the Python script inside a special stored procedure, [sp_execute_external_script](#).

After the script has been embedded in the stored procedure, any application that can make a stored procedure call can initiate execution of the Python code. Thereafter SQL Server manages code execution as summarized in the following diagram.



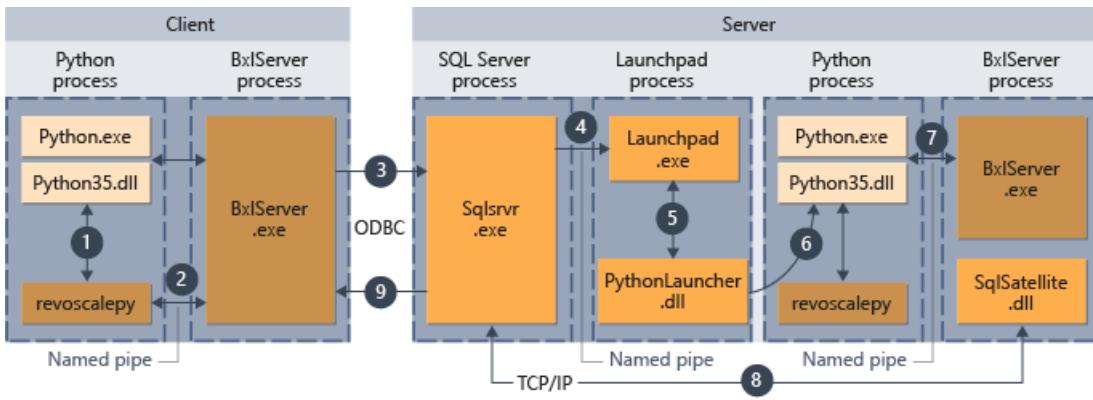
1. A request for the Python runtime is indicated by the parameter `@language='Python'` passed to the stored procedure. SQL Server sends this request to the Launchpad service.
2. The Launchpad service starts the appropriate launcher; in this case, `PythonLauncher`.
3. `PythonLauncher` starts the external `Python35` process.
4. `BxlServer` coordinates with the Python runtime to manage exchanges of data, and storage of working results.
5. SQL Satellite manages communications about related tasks and processes with SQL Server.
6. `BxlServer` uses SQL Satellite to communicate status and results to SQL Server.
7. SQL Server gets results and closes related tasks and processes.

Python scripts executed from a remote client

You can run Python scripts from a remote computer, such as a laptop, and have them execute in the context of the SQL Server computer, if these conditions are met:

- You design the scripts appropriately
- The remote computer has installed the extensibility libraries that are used by Machine Learning Services. The [revoscalepy](#) package is required to use remote compute contexts.

The following diagram summarizes the overall workflow when scripts are sent from a remote computer.



1. For functions that are supported in `revoscalepy`, the Python runtime calls a linking function, which in turn calls BxlServer.
2. BxlServer is included with Machine Learning Services (In-Database) and runs in a separate process from the Python runtime.
3. BxlServer determines the connection target and initiates a connection using ODBC, passing credentials supplied as part of the connection string in the Python script.
4. BxlServer opens a connection to the SQL Server instance.
5. When an external script runtime is called, the Launchpad service is invoked, which in turn starts the appropriate launcher: in this case, `PythonLauncher.dll`. Thereafter, processing of Python code is handled in a workflow similar to that when Python code is invoked from a stored procedure in T-SQL.
6. `PythonLauncher` makes a call to the instance of the Python that is installed on the SQL Server computer.
7. Results are returned to BxlServer.
8. SQL Satellite manages communication with SQL Server and cleanup of related job objects.
9. SQL Server passes results back to the client.

Next steps

[Architecture overview for Python in SQL Server](#)

Security overview for Python in SQL Server Machine Learning

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic describes the security architecture that is used to connect the SQL Server database engine and Python components. Examples of the security process are provided for two common scenarios: running Python in SQL Server using a stored procedure, and running Python with the SQL Server as the remote compute context.

Security overview

A SQL Server login or Windows user account is required to run Python script in SQL Server. These *security principals* are managed at the instance and database level, and identify users who have permission to connect to the database, read and write data, or create database objects such as tables or stored procedures. Additionally, users who run Python script must have permission to execute external script at the database level.

Even users who are using Python in an external tool must be mapped to a login or account in the database if the user needs to run Python code in-database, or access database objects and data. The same permissions are required whether the Python script is sent from a remote data science client or started using a T-SQL stored procedure.

For example, assume that you created a Python script that runs on your laptop, and you want to run that code on SQL Server. You must ensure that the following conditions are met:

- The database allows remote connections.
- The SQL login or Windows account that you used for database access has been added to the SQL Server at the instance level.
- The SQL login or Windows user must be granted the permission to execute external scripts. Generally, this permission can only be added by a database administrator.
- The SQL login or Window user must be added as a user, with appropriate permissions, in each database where the Python script performs any of these operations:
 - Retrieving data
 - Writing or updating data
 - Creating new objects, such as tables or stored procedures

After the login or Windows user account has been provisioned and given the necessary permissions, you can run Python code on SQL Server by using the data source objects provided by the **revoscalepy** library, or by calling a stored procedure that contains Python script.

Whenever a Python script is launched from SQL Server, the database engine security gets the security context of the user who started the job, and manages the mappings of the user or login to securable objects.

Therefore, all Python scripts that are initiated from a remote client must specify the login or user information as part of the connection string.

Interaction of SQL Server security and Launchpad security

When a Python script is executed in the context of the SQL Server computer, the SQL Server Trusted Launchpad service gets an available worker account (a local user account) from a pool of worker accounts established for

external processes, and uses that worker account to perform the related tasks.

For example, assume you launch a Python script under your Windows domain credentials. SQL Server gets your credentials and maps the task to an available Launchpad worker account, such as *SQLRUser01*.

NOTE

The name of the group of worker accounts is the same regardless of whether you are using R or Python. However, a separate group is created for each instance where you enable any external language.

After mapping to a worker account, SQL Server Trusted Launchpad creates a user token that is used to start processes.

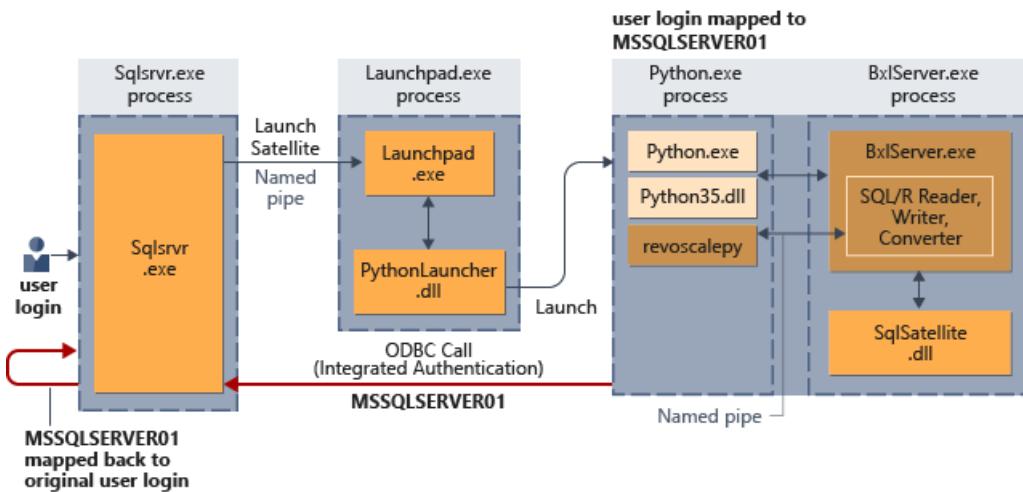
When all SQL Server operations are completed, the user worker account is marked as free and returned to the pool.

For more information about SQL Server Trusted Launchpad, see [Components in SQL Server to support Python integration](#).

Implied authentication

Implied authentication is the term used for the process under which SQL Server gets the user credentials and then executes all external script tasks on behalf of the users, assuming the user has the correct permissions in the database. Implied authentication is particularly important if the Python script needs to make an ODBC call outside the SQL Server database. For example, the code might retrieve a shorter list of factors from a spreadsheet or other source.

For such loopback calls to succeed, the group that contains the worker accounts, *SQLRUserGroup*, must have "Allow Log on locally" permissions. By default, this right is given to all new local users, but in some organizations stricter group policies might be enforced.



Security of worker accounts

The mapping of an external Windows user or valid SQL login to a worker account is valid only for the lifetime of the SQL stored procedure that executes the Python script.

Parallel queries from the same login are mapped to the same user worker account.

The directories used for the processes are managed by the SQL Server Trusted Launchpad, and directories are access-restricted. For Python, PythonLauncher performs this task. Each individual worker account is restricted to its own folder, and cannot access files in folders above its own level. However, the worker account can read, write, or delete children under the session working folder that was created.

For more information about how to change the number of worker accounts, account names, or account

passwords, see [Modify the user account pool for SQL Server machine learning](#).

Security isolation for multiple external scripts

The isolation mechanism is based on physical user accounts. As satellite processes are started for a specific language runtime, each satellite task uses the worker account specified by the SQL Server Trusted Launchpad. If a task requires multiple satellites, for example, in the case of parallel queries, a single worker account is used for all related tasks.

No worker account can see or manipulate files used by other worker accounts.

If you are an administrator on the computer, you can view the directories created for each process. Each directory is identified by its session GUID.

See Also

[Architecture overview](#)

Install SQL Server 2017 Machine Learning Services (In-Database) on Windows

6/28/2018 • 14 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The Machine Learning Services component of SQL Server adds in-database predictive analytics, statistical analysis, visualization, and machine learning algorithms. Function libraries are available in R and Python and run as external script on a database engine instance.

This article explains how to install the machine learning component by running the SQL Server setup wizard, and following the on-screen prompts.

Pre-install checklist

- SQL Server 2017 setup is required if you want to install Machine Learning Services with language support for R, Python, or both. If instead you have SQL Server 2016 installation media, you can install [SQL Server 2016 R Services \(In-Database\)](#) to get R language support.
- A database engine instance is required. You cannot install just R or Python features, although you can add them incrementally to an existing instance.
- Do not install Machine Learning Services on a failover cluster. The security mechanism used for isolating R and Python processes is not compatible with a Windows Server failover cluster environment.
- Do not install Machine Learning Services on a domain controller. The Machine Learning Services portion of setup will fail.
- Do not install **Shared Features > Machine Learning Server (Standalone)** on the same computer running an in-database instance. A standalone server will compete for the same resources, undermining the performance of both installations.
- Side-by-side installation with other versions of R and Python is supported but not recommended. It's supported because SQL Server instance uses its own copies of the open-source R and Anaconda distributions. But it's not recommended because running code that uses R and Python on the SQL Server computer outside SQL Server can lead to various problems:
 - You use a different library and different executable, and get different results, than you do when you are running in SQL Server.
 - R and Python scripts running in external libraries cannot be managed by SQL Server, leading to resource contention.

IMPORTANT

After setup is complete, be sure to complete the post-configuration steps described in this article. These steps include enabling SQL Server to use external scripts, and adding accounts required for SQL Server to run R and Python jobs on your behalf. Configuration changes generally require a restart of the instance, or a restart of the Launchpad service.

Get the installation media

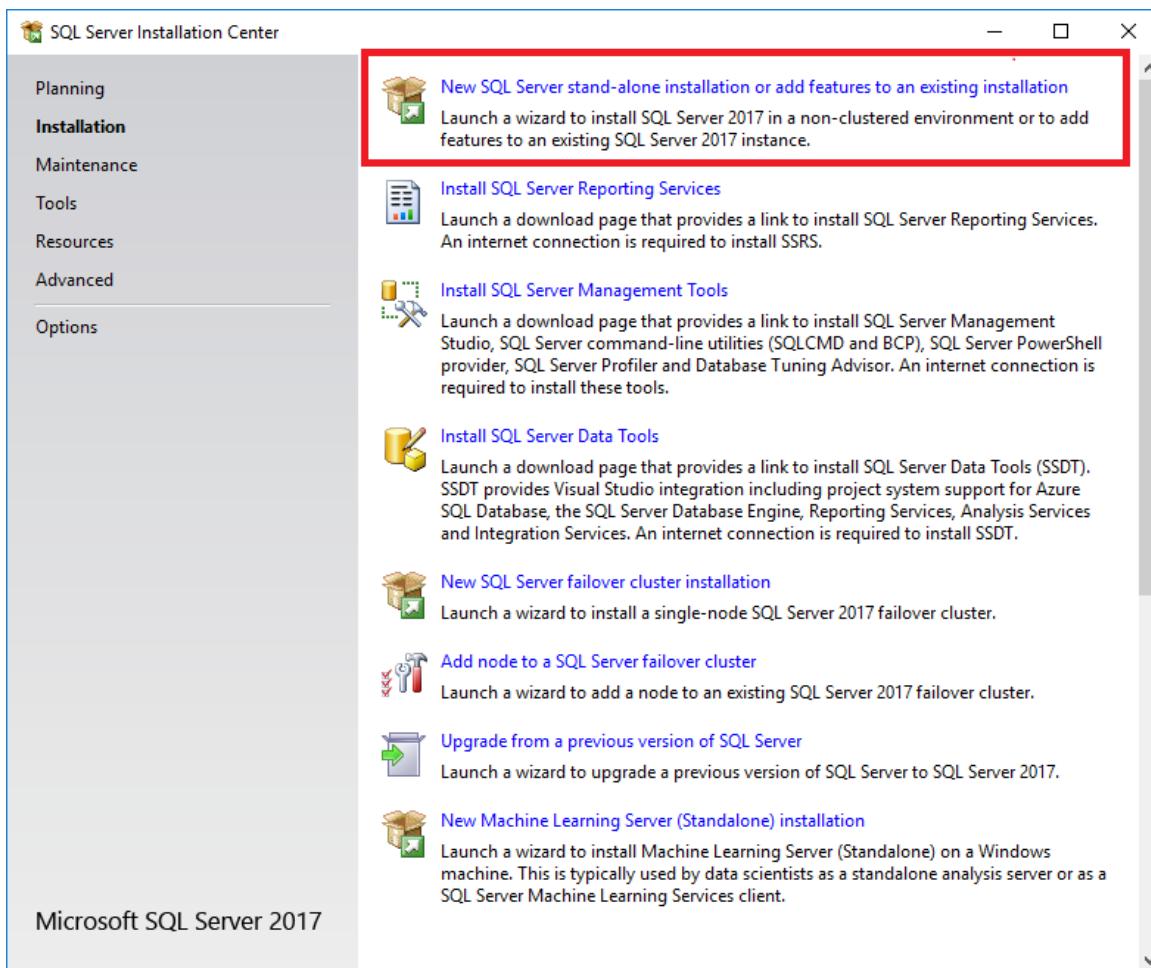
The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express Editions** are licensed for production use. For Enterprise and Standard Editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- **Free editions** are available at [SQL Server Downloads](#).

Run Setup

For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

1. Start the setup wizard for SQL Server 2017. You can download
2. On the **Installation** tab, select **New SQL Server stand-alone installation or add features to an existing installation**.



3. On the **Feature Selection** page, select these options:

- **Database Engine Services**

To use R and Python with SQL Server, you must install an instance of the database engine. You can use either a default or a named instance.

- **Machine Learning Services (In-Database)**

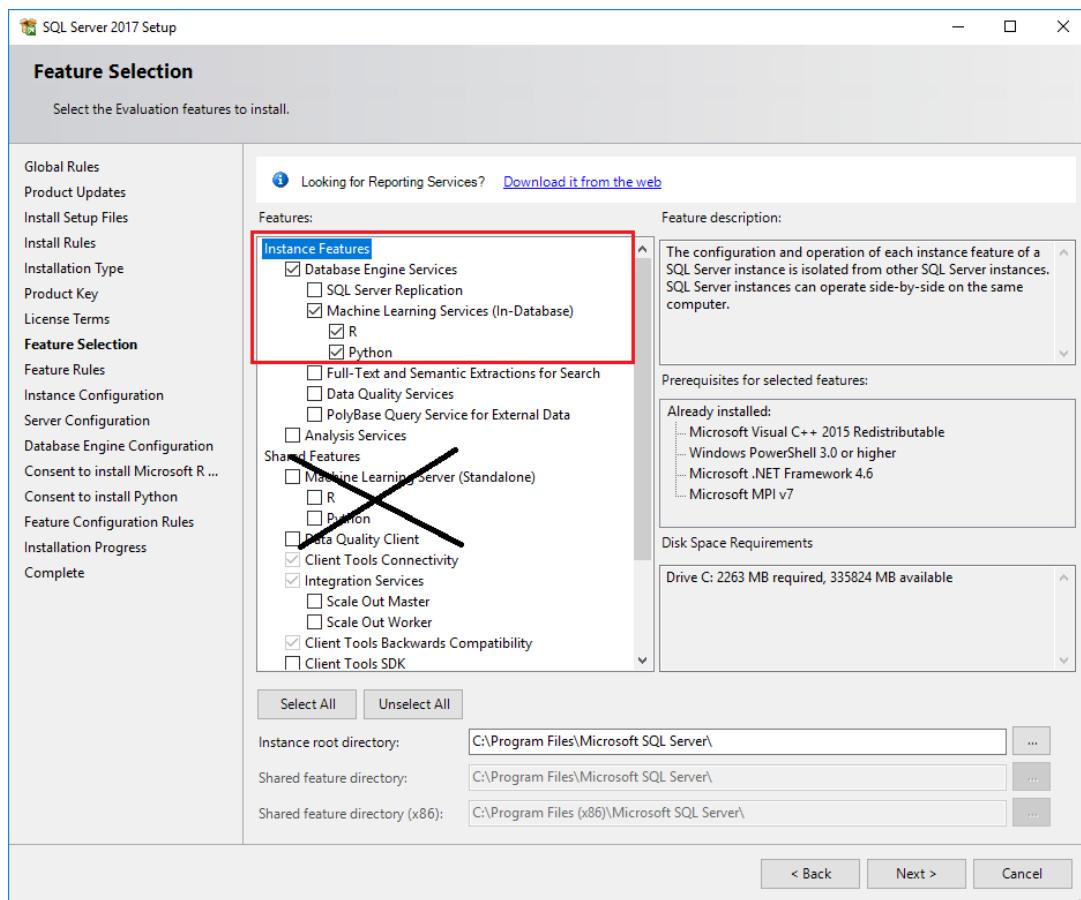
This option installs the database services that support R and Python script execution.

- **R**

Check this option to add the Microsoft R packages, interpreter, and open-source R.

- **Python**

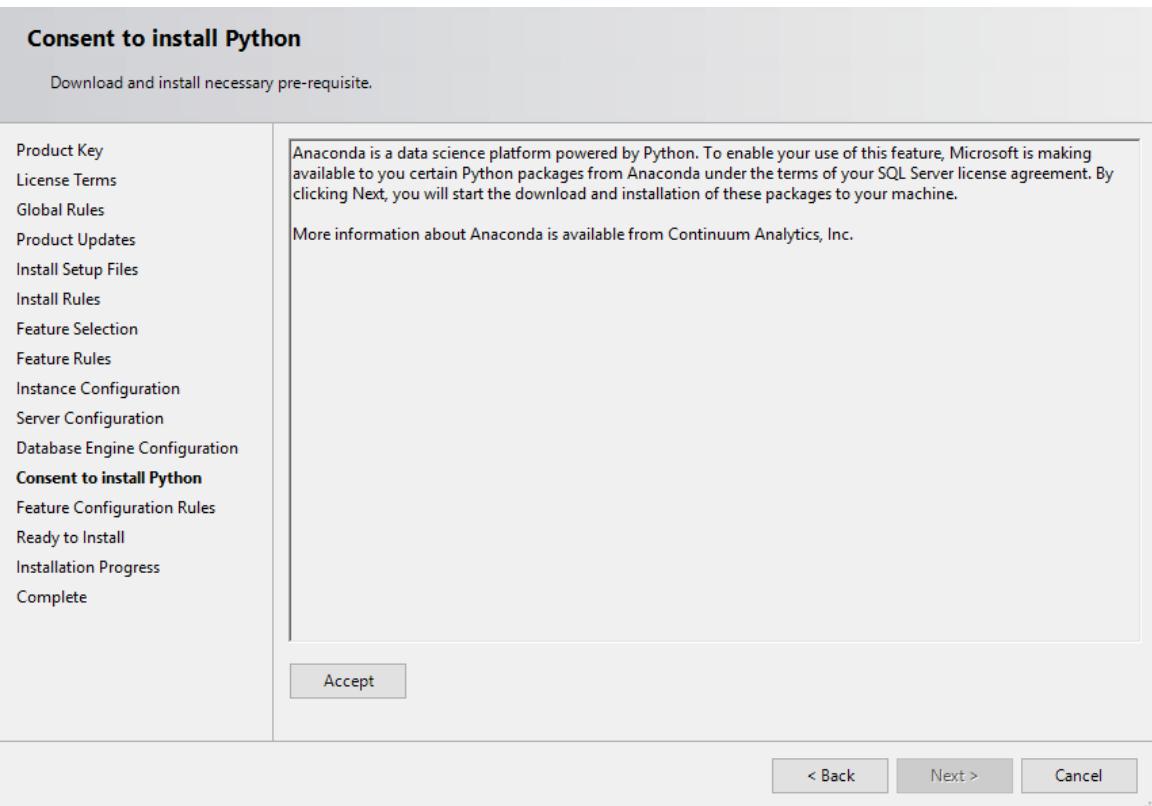
Check this option to add the Microsoft Python packages, the Python 3.5 executable, and select libraries from the Anaconda distribution.



NOTE

Do not select the option for **Machine Learning Server (Standalone)**. The option to install Machine Learning Server under **Shared Features** is intended for use on a separate computer.

4. On the **Consent to Install R** page, select **Accept**. This license agreement covers Microsoft R Open, which includes a distribution of the open source R base packages and tools, together with enhanced R packages and connectivity providers from the Microsoft development team.
5. On the **Consent to Install Python** page, select **Accept**. The Python open source licensing agreement also covers Anaconda and related tools, plus some new Python libraries from the Microsoft development team.



NOTE

If the computer you are using does not have internet access, you can pause setup at this point to download the installers separately. For more information, see [Install machine learning components without internet access](#).

- Select **Accept**, wait until the **Next** button becomes active, and then select **Next**.
6. On the **Ready to Install** page, verify that these selections are included, and select **Install**.

- Database Engine Services
- Machine Learning Services (In-Database)
- R or Python, or both

Note of the location of the folder under the path `..\Setup_Bootstrap\Log` where the configuration files are stored. When setup is complete, you can review the installed components in the Summary file.

Restart the service

When the installation is complete, restart the database engine before continuing to the next, enabling script execution.

Restarting the service also automatically restarts the related SQL Server Trusted Launchpad service.

You can restart the service using the right-click **Restart** command for the instance in SSMS, or by using the **Services** panel in Control Panel, or by using [SQL Server Configuration Manager](#).

Enable external script execution

1. Open SQL Server Management Studio.

TIP

You can download and install the appropriate version from this page: [Download SQL Server Management Studio \(SSMS\)](#).

You can also try out the preview release of [SQL Operations Studio](#), which supports administrative tasks and queries against SQL Server.

2. Connect to the instance where you installed Machine Learning Services, click **New Query** to open a query window, and run the following command:

```
sp_configure
```

The value for the property, `external scripts enabled`, should be **0** at this point. That is because the feature is turned off by default. The feature must be explicitly enabled by an administrator before you can run R or Python scripts.

3. To enable the external scripting feature, run the following statement:

```
EXEC sp_configure 'external scripts enabled', 1  
RECONFIGURE WITH OVERRIDE
```

If you have already enabled the feature for the R language, don't run reconfigure a second time for Python. The underlying extensibility platform supports both languages.

4. Restart the SQL Server service for the SQL Server instance. Restarting the SQL Server service also automatically restarts the related SQL Server Trusted Launchpad service.

You can restart the service using the right-click **Restart** command for the instance in SSMS, or by using the **Services** panel in Control Panel, or by using [SQL Server Configuration Manager](#).

Verify installation

Use the following steps to verify that all components used to launch external script are running.

1. In SQL Server Management Studio, open a new query window, and run the following command:

```
EXEC sp_configure 'external scripts enabled'
```

The **run_value** should now be set to 1.

2. Open the **Services** panel or SQL Server Configuration Manager, and verify **SQL Server Launchpad service** is running. You should have one service for every database engine instance that has R or Python installed. Restart the service if it is not running. For more information, see [Components to support Python integration](#).
3. If Launchpad is running, you should be able to run simple R and Python scripts to verify that external scripting runtimes can communicate with SQL Server.

Open a new **Query** window in SQL Server Management Studio, and then run a script such as the following:

- For R

```
EXEC sp_execute_external_script @language =N'R',
@script=N'
OutputDataSet <- InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS (([hello] int not null));
GO
```

- For Python

```
EXEC sp_execute_external_script @language =N'Python',
@script=N'
OutputDataSet = InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS (([hello] int not null));
GO
```

Results

The script can take a little while to run, the first time the external script runtime is loaded. The results should be something like this:

HELLO
1

NOTE

Columns or headings used in the Python script are not returned, by design. To add column names for your output, you must specify the schema for the return data set. Do this by using the WITH RESULTS parameter of the stored procedure, naming the columns and specifying the SQL data type.

For example, you can add the following line to generate an arbitrary column name:

```
WITH RESULT SETS ((Col1 AS int))
```

Additional configuration

If the external script verification step was successful, you can run Python commands from SQL Server Management Studio, Visual Studio Code, or any other client that can send T-SQL statements to the server.

If you got an error when running the command, review the additional configuration steps in this section. You might need to make additional appropriate configurations to the service or database.

Common scenarios that require additional changes include:

- [Configure Windows firewall for in-bound connections](#)
- [Enable additional network protocols](#)
- [Enable remote connections](#)
- [Extend built-in permissions to remote users](#)
- [Grant permission to run external scripts](#)
- [Grant access to individual databases](#)

NOTE

Not all the listed changes are required, and none might be required. Requirements depend on your security schema, where you installed SQL Server, and how you expect users to connect to the database and run external scripts. Additional troubleshooting tips can be found here: [Upgrade and installation FAQ](#)

Enable implied authentication for Launchpad account group

During setup, a number of new Windows user accounts are created for the purpose of running tasks under the security token of the SQL Server Trusted Launchpad service. When a user sends a Python or R script from an external client, SQL Server activates an available worker account. Then it maps it to the identity of the calling user, and runs the script on behalf of the user.

This is called *implied authentication*, and is a service of the database engine. This service supports secure execution of external scripts in SQL Server 2016 and SQL Server 2017.

You can view these accounts in the Windows user group, **SQLRUserGroup**. By default, 20 worker accounts are created, which is usually more than enough for running external script jobs.

IMPORTANT

The worker group is named **SQLRUserGroup** regardless of whether you installed R or Python. There is a single group for each instance.

If you need to run scripts from a remote data science client, and you are using Windows authentication, there are additional considerations. These worker accounts must be given permission to sign in to the SQL Server instance on your behalf.

1. In SQL Server Management Studio, in Object Explorer, expand **Security**. Then right-click **Logins**, and select **New Login**.
2. In the **Login - New** dialog box, select **Search**.
3. Select **Object Types**, and select **Groups**. Clear everything else.
4. In **Enter the object name to select**, type *SQLRUserGroup*, and select **Check Names**.
5. The name of the local group associated with the instance's Launchpad service should resolve to something like *instancename\SQLRUserGroup*. Select **OK**.
6. By default, the group is assigned to the **public** role, and has permission to connect to the database engine.
7. Select **OK**.

NOTE

If you use a **SQL login** for running scripts in a SQL Server compute context, this extra step is not required.

Give users permission to run external scripts

If you installed SQL Server yourself, and you are running R or Python scripts in your own instance, you typically execute scripts as an administrator. Thus, you have implicit permission over various operations and all data in the database.

Most users, however, do not have such elevated permissions. For example, users in an organization who use SQL logins to access the database generally do not have elevated permissions. Therefore, for each user who is using R or Python, you must grant users of Machine Learning Services the permission to run external scripts in each database where the language is used. Here's how:

```
USE <database_name>
GO
GRANT EXECUTE ANY EXTERNAL SCRIPT TO [UserName]
```

NOTE

Permissions are not specific to the supported script language. In other words, there are not separate permission levels for R script versus Python script. If you need to maintain separate permissions for these languages, install R and Python on separate instances.

Give your users read, write, or data definition language (DDL) permissions to databases

While a user is running scripts, the user might need to read data from other databases. The user might also need to create new tables to store results, and write data into tables.

For each Windows user account or SQL login that is running R or Python scripts, ensure that it has the appropriate permissions on the specific database: `db_datareader`, `db_datawriter`, or `db_ddladmin`.

For example, the following Transact-SQL statement gives the SQL login *MySQLLogin* the rights to run T-SQL queries in the *ML_Samples* database. To run this statement, the SQL login must already exist in the security context of the server.

```
USE ML_Samples
GO
EXEC sp_addrolemember 'db_datareader', 'MySQLLogin'
```

For more information about the permissions included in each role, see [Database-level roles](#).

Create an ODBC data source for the instance on your data science client

You might create a machine learning solution on a data science client computer. If you need to run code by using the SQL Server computer as the compute context, you have two options: access the instance by using a SQL login, or by using a Windows account.

- For SQL logins: Ensure that the login has appropriate permissions on the database where you are reading data. You can do this by adding *Connect to* and *SELECT* permissions, or by adding the login to the `db_datareader` role. To create objects, assign `DDL_admin` rights. If you must save data to tables, add to the `db_datawriter` role.
- For Windows authentication: You might need to create an ODBC data source on the data science client that specifies the instance name and other connection information. For more information, see [ODBC data source administrator](#).

Suggested optimizations

Now that you have everything working, you might also want to optimize the server to support machine learning, or install pretrained models.

Add more worker accounts

If you expect many users to be running scripts concurrently, you can increase the number of worker accounts that are assigned to the Launchpad service. For more information, see [Modify the user account pool for SQL Server Machine Learning Services](#).

Optimize the server for script execution

The default settings for SQL Server setup are intended to optimize the balance of the server for a variety of services that are supported by the database engine, which might include extract, transform, and load (ETL)

processes, reporting, auditing, and applications that use SQL Server data. Therefore, under the default settings, you might find that resources for machine learning are sometimes restricted or throttled, particularly in memory-intensive operations.

To ensure that machine learning jobs are prioritized and resourced appropriately, we recommend that you use SQL Server Resource Governor to configure an external resource pool. You might also want to change the amount of memory that's allocated to the SQL Server database engine, or increase the number of accounts that run under the SQL Server Trusted Launchpad service.

- To configure a resource pool for managing external resources, see [Create an external resource pool](#).
- To change the amount of memory reserved for the database, see [Server memory configuration options](#).
- To change the number of R accounts that can be started by SQL Server Trusted Launchpad, see [Modify the user account pool for machine learning](#).

If you are using Standard Edition and do not have Resource Governor, you can use Dynamic Management Views (DMVs) and Extended Events, as well as Windows event monitoring, to help manage the server resources. For more information, see [Monitoring and managing R Services](#) and [Monitoring and managing Python Services](#).

Install additional R packages

The R solutions you create for SQL Server can call basic R functions, functions from the proprietary packages installed with SQL Server, and third-party R packages compatible with the version of open-source R installed by SQL Server.

Packages that you want to use from SQL Server must be installed in the default library that is used by the instance. If you have a separate installation of R on the computer, or if you installed packages to user libraries, you won't be able to use those packages from T-SQL.

The process for installing and managing R packages is different in SQL Server 2016 and SQL Server 2017. In SQL Server 2016, a database administrator must install R packages that users need. In SQL Server 2017, you can set up user groups to share packages on a per-database level, or configure database roles to enable users to install their own packages. For more information, see [Install new R packages in SQL Server](#).

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Tutorial: Run Python in T-SQL](#)
- [Tutorial: In-database analytics for Python developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Install SQL Server 2017 Machine Learning Server (Standalone) on Windows

6/1/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server setup includes the option to install a machine learning server that runs outside of SQL Server. This option might be useful if you need to develop high-performance machine learning solutions that can use remote compute contexts, switching interchangeably between the local server and a remote Machine Learning Server on a Spark cluster or on another SQL Server instance.

This article describes how to use SQL Server setup to install the standalone version of **SQL Server 2017 Machine Learning Server**.

Pre-install checklist

SQL Server 2017 is required. If you have SQL Server 2016, please install [SQL Server 2016 R Server \(Standalone\)](#) instead.

If you installed a previous version, such as SQL Server 2016 R Server (Standalone) or Microsoft R Server, uninstall the existing installation before continuing.

Get the installation media

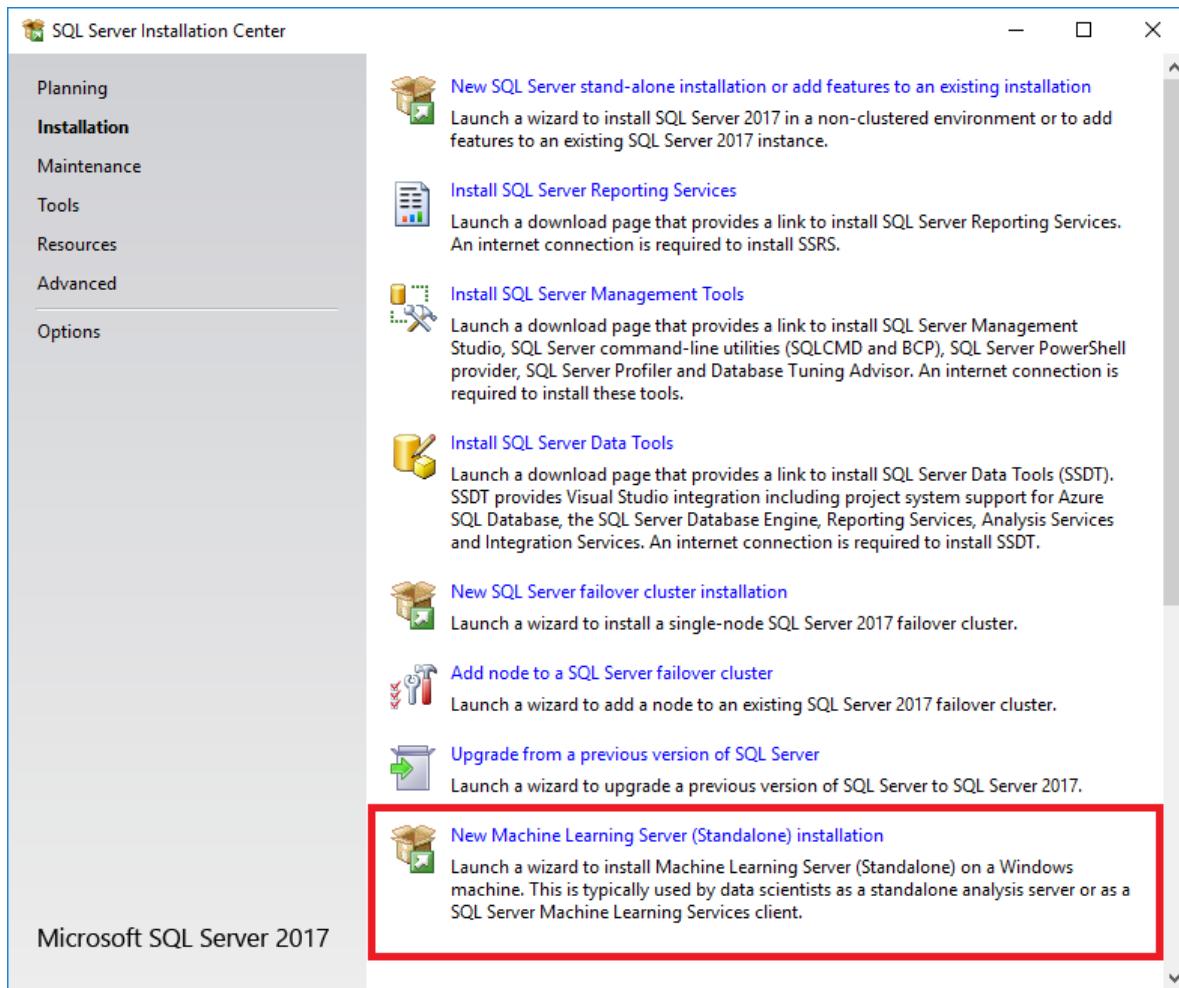
The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express Editions** are licensed for production use. For Enterprise and Standard Editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- **Free editions** are available at [SQL Server Downloads](#).

Run Setup

For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

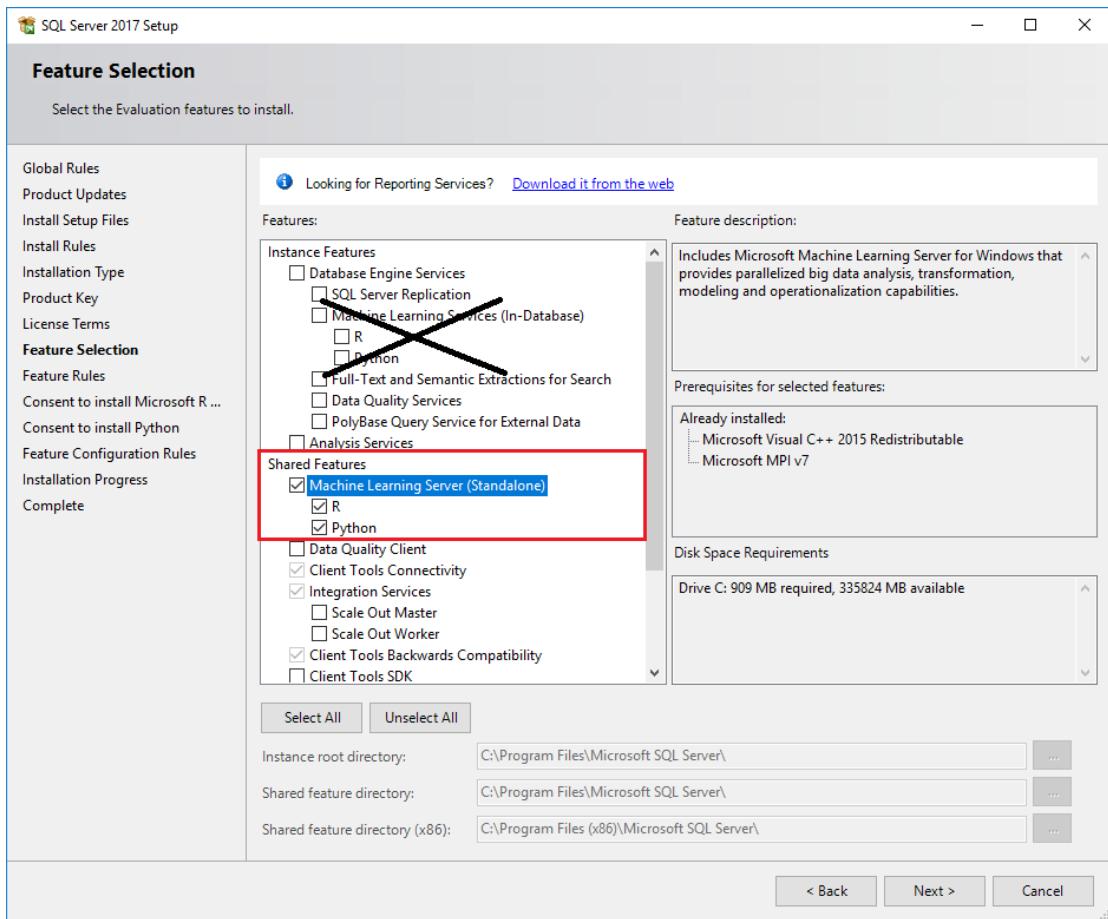
1. Start the setup wizard for SQL Server 2017.
2. Click the **Installation** tab, and select **New Machine Learning Server (Standalone) installation**.



3. After the rules check is complete, accept SQL Server licensing terms, and select a new installation.

4. On the **Feature Selection** page, the following options should already be selected:

- Microsoft Machine Learning Server (Standalone)
- R and Python are both selected by default. You can deselect either language, but we recommend that you install at least one of the supported languages.



All other options should be ignored.

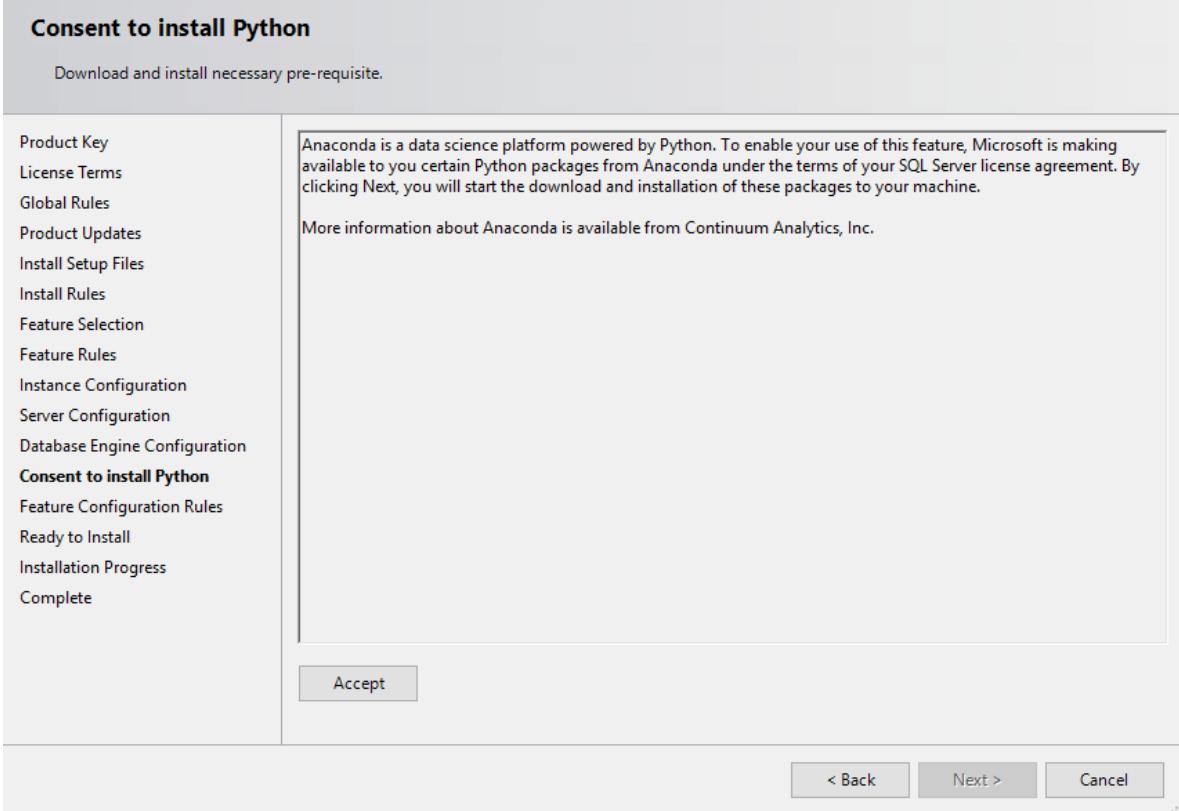
NOTE

Avoid installing the **Shared Features** if the computer already has Machine Learning Services installed for SQL Server in-database analytics. This creates duplicate libraries.

Also, whereas R or Python scripts running in SQL Server are managed by SQL Server so as not to conflict with memory used by other database engine services, the standalone machine learning server has no such constraints, and can interfere with other database operations. Finally, remote access via RDP session, which is often used for operationalization, is typically blocked by database administrators.

For these reasons, we generally recommend that you install Machine Learning Server (Standalone) on a separate computer from SQL Server Machine Learning Services.

5. Accept the license terms for downloading and installing the machine learning components. If you install both languages, a separate licensing agreement is required for Microsoft R and for Python.



Installation of these components, and any prerequisites they might require, might take a while. When the **Accept** button becomes unavailable, you can click **Next**.

6. On the **Ready to Install** page, verify your selections, and click **Install**.

Default installation folders

When you install R Server or Machine Learning Server using SQL Server setup, the R libraries are installed in a folder associated with the SQL Server version that you used for setup. In this folder, you will also find sample data, documentation for the R base packages, and documentation of the R tools and runtime.

However, if you install using the separate Windows installer, or if you upgrade using the separate Windows installer, the R libraries are installed in a different folder.

Just for reference, if you have installed an instance of SQL Server with R Services (In-Database) or Machine Learning Services (In-Database), and that instance is on the same computer, the R libraries and tools are installed by default in a different folder.

The following table lists the paths for each installation.

VERSION	INSTALLATION METHOD	DEFAULT FOLDER
SQL Server 2017 Machine Learning Server (Standalone)	SQL Server 2017 setup wizard	C:\Program Files\Microsoft SQL Server\140\R_SERVER C:\Program Files\Microsoft SQL Server\140\PYTHON_SERVER
Microsoft Machine Learning Server (Standalone)	Windows standalone installer	C:\Program Files\Microsoft\ML Server\R_SERVER C:\Program Files\Microsoft\ML Server\PYTHON_SERVER

VERSION	INSTALLATION METHOD	DEFAULT FOLDER
SQL Server 2017 Machine Learning Services (In-Database)	SQL Server 2017 setup wizard, with R language option	C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\R_SERVICES C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\PYTHON_SERVICES
SQL Server 2016 R Server (Standalone)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\130\R_SERVER
SQL Server 2016 R Services (In-Database)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\MSSQL13.<instance_name>\R_SERVICES

Development tools

A development IDE is not installed as part of setup. Additional tools are not required, as all the standard tools are included that would be provided with a distribution of R or Python.

We recommend that you try the new release of R Tools for Visual Studio or [Python for Visual Studio](#). Visual Studio supports both R and Python, as well as database development tools, connectivity with SQL Server, and BI tools. However, you can use any preferred development environment, including RStudio.

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Tutorial: Run Python in T-SQL](#)
- [Tutorial: In-database analytics for Python developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Install SQL Server 2016 R Services (In-Database)

5/30/2018 • 13 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article explains how to install and configure **SQL Server 2016 R Services (In-Database)**. If you have SQL Server 2016, install this feature to enable execution of R code in SQL Server.

Pre-install checklist

- SQL Server 2016 setup is required if you want to install R Services. If instead you have SQL Server 2017 installation media, you should install [SQL Server 2017 Machine Learning Services \(In-Database\)](#) to get R integration for that version of SQL Server.
- A database engine instance is required. You cannot install just R, although you can add it incrementally to an existing instance.
- Do not install R Services on a failover cluster. The security mechanism used for isolating R processes is not compatible with a Windows Server failover cluster environment.
- Do not install R Services on a domain controller. The R Services portion of setup will fail.
- Do not install **Shared Features > R Server (Standalone)** on the same computer running an in-database instance.
- Side-by-side installation with other versions of R and Python are possible because the SQL Server instance uses its own copies of the open-source R and Anaconda distributions. However, running code that uses R and Python on the SQL Server computer outside SQL Server can lead to various problems:
 - You use a different library and different executable, and get different results, than you do when you are running in SQL Server.
 - R and Python scripts running in external libraries cannot be managed by SQL Server, leading to resource contention.

If you used any earlier versions of the Revolution Analytics development environment or the RevoScaleR packages, or if you installed any pre-release versions of SQL Server 2016, you must uninstall them. Running older and newer versions of RevoScaleR and other proprietary packages is not supported. For help removing previous versions, see [Upgrade and Installation FAQ for SQL Server Machine Learning Services](#).

IMPORTANT

After setup is complete, be sure to complete the additional post-configuration steps described in this article. These steps include enabling SQL Server to use external scripts, and adding accounts required for SQL Server to run R jobs on your behalf. Configuration changes generally require a restart of the instance, or a restart of the Launchpad service.

Get the installation media

The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express Editions** are licensed for production use. For Enterprise and Standard Editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).

- **Free editions** are available at [SQL Server Downloads](#).

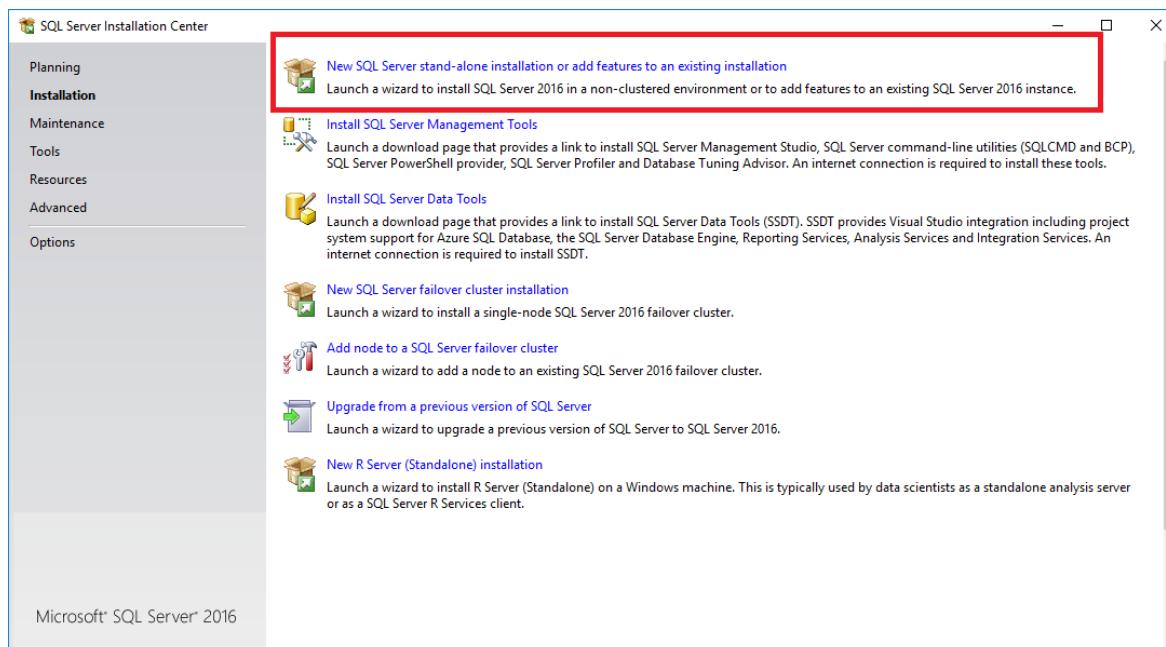
Install patch requirement

Microsoft has identified a problem with the specific version of Microsoft VC++ 2013 Runtime binaries that are installed as a prerequisite by SQL Server. If this update to the VC runtime binaries is not installed, SQL Server may experience stability issues in certain scenarios. Before you install SQL Server follow the instructions at [SQL Server Release Notes](#) to see if your computer requires a patch for the VC runtime binaries.

Run Setup

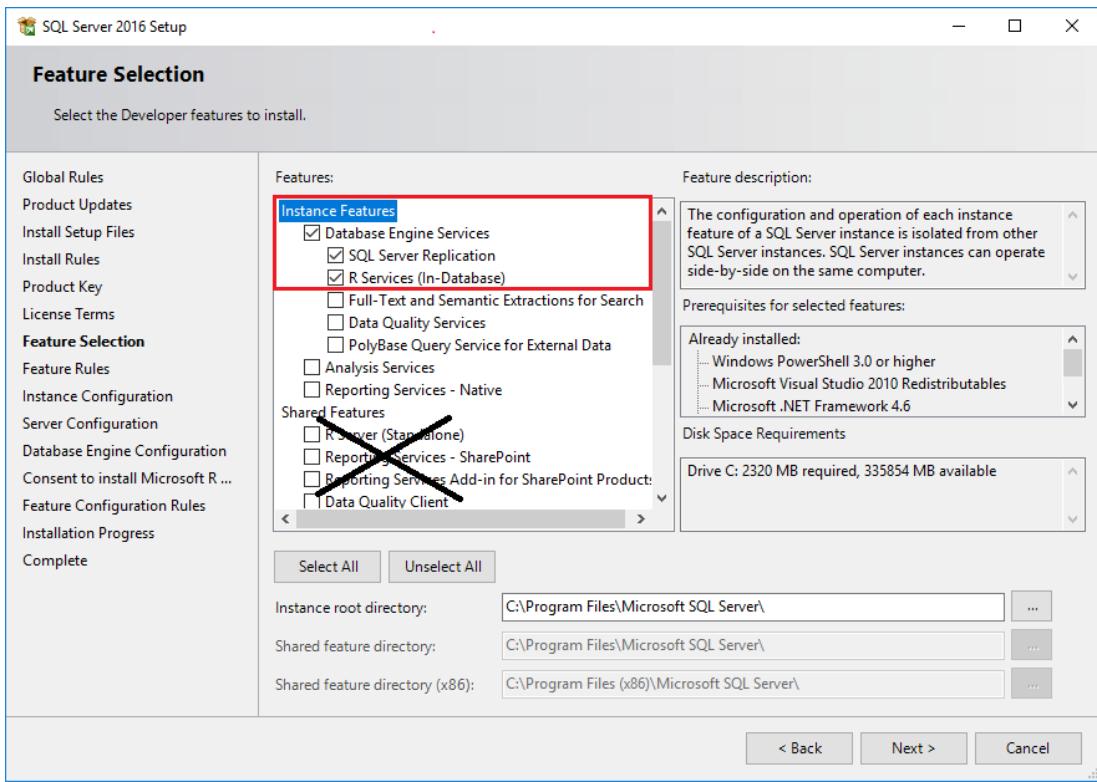
For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

1. Start the setup wizard for SQL Server 2016.
2. On the **Installation** tab, select **New SQL Server stand-alone installation or add features to an existing installation**.



3. On the **Feature Selection** page, select the following options:

- Select **Database Engine Services**. The database engine is required in each instance that uses machine learning.
- Select **R Services (In-Database)**. Installs support for in-database use of R.



IMPORTANT

Do not install R Server and R Services at the same time. You would ordinarily install R Server (Standalone) to create an environment that a data scientist or developer uses to connect to SQL Server and deploy R solutions. Therefore, there is no need to install both on the same computer.

4. On the **Consent to Install Microsoft R Open** page, click **Accept**.

This license agreement is required to download Microsoft R Open, which includes a distribution of the open source R base packages and tools, together with enhanced R packages and connectivity providers from the Microsoft R development team.

5. After you have accepted the license agreement, there is a brief pause while the installer is prepared. Click **Next** when the button becomes available.

6. On the **Ready to Install** page, verify that the following items are included, and then select **Install**.

- Database Engine Services
- R Services (In-Database)

Note of the location of the folder under the path `..\Setup_Bootstrap\Log` where the configuration files are stored. When setup is complete, you can review the installed components in the Summary file.

7. When the installation is complete, restart your computer.

Enable external script execution

1. Open SQL Server Management Studio.

TIP

You can download and install the appropriate version from this page: [Download SQL Server Management Studio \(SSMS\)](#).

You can also try out the preview release of [SQL Operations Studio](#), which supports administrative tasks and queries against SQL Server.

2. Connect to the instance where you installed Machine Learning Services, click **New Query** to open a query window, and run the following command:

```
sp_configure
```

The value for the property, `external scripts enabled`, should be **0** at this point. That is because the feature is turned off by default. The feature must be explicitly enabled by an administrator before you can run R or Python scripts.

3. To enable the external scripting feature, run the following statement:

```
EXEC sp_configure 'external scripts enabled', 1  
RECONFIGURE WITH OVERRIDE
```

Restart the service

When the installation is complete, restart the database engine before continuing to the next, enabling script execution.

Restarting the service also automatically restarts the related SQL Server Trusted Launchpad service.

You can restart the service using the right-click **Restart** command for the instance in SSMS, or by using the **Services** panel in Control Panel, or by using [SQL Server Configuration Manager](#).

Verify installation

Use the following steps to verify that all components used to launch external script are running.

1. In SQL Server Management Studio, open a new query window, and run the following command:

```
EXEC sp_configure 'external scripts enabled'
```

The **run_value** should now be set to 1.

2. Open the **Services** panel or SQL Server Configuration Manager, and verify **SQL Server Launchpad service** is running. You should have one service for every database engine instance that has R or Python installed. Restart the service if it is not running. For more information, see [Components to support Python integration](#).
3. If Launchpad is running, you should be able to run simple R to verify that external scripting runtimes can communicate with SQL Server.

Open a new **Query** window in SQL Server Management Studio, and then run a script such as the following:

```
EXEC sp_execute_external_script @language =N'R',
@script=N'
OutputDataSet <- InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS (([hello] int not null));
GO
```

The script can take a little while to run, the first time the external script runtime is loaded. The results should be something like this:

HELLO
1

Additional configuration

If the external script verification step was successful, you can run Python commands from SQL Server Management Studio, Visual Studio Code, or any other client that can send T-SQL statements to the server.

If you got an error when running the command, review the additional configuration steps in this section. You might need to make additional appropriate configurations to the service or database.

Common scenarios that require additional changes include:

- [Configure Windows firewall for in-bound connections](#)
- [Enable additional network protocols](#)
- [Enable remote connections](#)
- [Extend built-in permissions to remote users](#)
- [Grant permission to run external scripts](#)
- [Grant access to individual databases](#)

NOTE

Not all the listed changes are required, and none might be required. Requirements depend on your security schema, where you installed SQL Server, and how you expect users to connect to the database and run external scripts. Additional troubleshooting tips can be found here: [Upgrade and installation FAQ](#)

Enable implied authentication for the Launchpad account group

During setup, some new Windows user accounts are created for running tasks under the security token of the SQL Server Trusted Launchpad service. When a user sends an R script from an external client, SQL Server activates an available worker account, maps it to the identity of the calling user, and runs the R script on behalf of the user. This new service of the database engine supports the secure execution of external scripts, called *implied authentication*.

You can view these accounts in the Windows user group **SQLRUserGroup**. By default, 20 worker accounts are created, which is usually more than enough for running R jobs.

However, if you need to run R scripts from a remote data science client and are using Windows authentication, you must grant these worker accounts permission to sign in to the SQL Server instance on your behalf.

1. In SQL Server Management Studio, in Object Explorer, expand **Security**, right-click **Logins**, and select **New Login**.
2. In the **Login - New** dialog box, select **Search**.

3. Select the **Object Types** and **Groups** check boxes, and clear all other check boxes.
4. Click **Advanced**, verify that the location to search is the current computer, and then click **Find Now**.
5. Scroll through the list of group accounts on the server until you find one beginning with `SQLRUserGroup`.

- The name of the group that's associated with the Launchpad service for the *default instance* is always just **SQLRUserGroup**. Select this account only for the default instance.
 - If you are using a *named instance*, the instance name is appended to the default name, `SQLRUserGroup`. Hence, if your instance is named "MLTEST", the default user group name for this instance would be **SQLRUserGroupMLTest**.
6. Click **OK** to close the advanced search dialog box, and verify that you've selected the correct account for the instance. Each instance can use only its own Launchpad service and the group created for that service.
 7. Click **OK** once more to close the **Select User or Group** dialog box.
 8. In the **Login - New** dialog box, click **OK**. By default, the login is assigned to the **public** role and has permission to connect to the database engine.

Give users permission to run external scripts

NOTE

If you use a SQL login for running R scripts in a SQL Server compute context, this step is not required.

If you installed SQL Server in your own instance, you are usually executing scripts as an administrator, or at least as a database owner, and thus have implicit permissions for various operations, all data in the database, and the ability to install new packages as needed.

However, in an enterprise scenario, most users, including users who access the database by using SQL logins, do not have such elevated permissions. Therefore, for each user who will be running R scripts, you must grant the user permissions to run scripts in each database where external scripts will be used.

```
USE <database_name>
GO
GRANT EXECUTE ANY EXTERNAL SCRIPT TO [UserName]
```

TIP

Need help with setup? Not sure you have run all the steps? Use these custom reports to check installation status and run additional steps.

[Monitor Machine Learning Services using Custom Reports.](#)

Give your users read, write, or DDL permissions to the database

The user account that is used to run R might need to read data from other databases, create new tables to store results, and write data into tables. Therefore, for each user who will be executing R scripts, ensure that the user has the appropriate permissions on the database: `db_datareader`, `db_datawriter`, or `db_ddladmin`.

For example, the following Transact-SQL statement gives the SQL login `MySQLLogin` the rights to run T-SQL queries in the `RSamples` database. To run this statement, the SQL login must already exist in the security context of the server.

```
USE RSamples
GO
EXEC sp_addrolemember 'db_datareader', 'MySQLLogin'
```

For more information about the permissions included in each role, see [Database-level roles](#).

Create an ODBC data source for the instance on your data science client

If you create an R solution on a data science client computer and need to run code by using the SQL Server computer as the compute context, you can use either a SQL login or integrated Windows authentication.

- For SQL logins: Ensure that the login has appropriate permissions on the database where you will be reading data. You can do so by adding *Connect to* and *SELECT* permissions, or by adding the login to the *db_datareader* role. For logins that need to create objects, add *DDL_admin* rights. For logins that must save data to tables, add the login to the *db_datawriter* role.
- For Windows authentication: You might need to configure an ODBC data source on the data science client that specifies the instance name and other connection information. For more information, see [ODBC data source administrator](#).

Suggested optimizations

Now that you have everything working, you might also want to optimize the server to support machine learning, or install pretrained models.

Add more worker accounts

If you think you might use R heavily, or if you expect many users to be running scripts concurrently, you can increase the number of worker accounts that are assigned to the Launchpad service. For more information, see [Modify the user account pool for SQL Server Machine Learning Services](#).

Optimize the server for external script execution

The default settings for SQL Server setup are intended to optimize the balance of the server for a variety of services that are supported by the database engine, which might include extract, transform, and load (ETL) processes, reporting, auditing, and applications that use SQL Server data. Therefore, under the default settings, you might find that resources for machine learning are sometimes restricted or throttled, particularly in memory-intensive operations.

To ensure that machine learning jobs are prioritized and resourced appropriately, we recommend that you use SQL Server Resource Governor to configure an external resource pool. You might also want to change the amount of memory that's allocated to the SQL Server database engine, or increase the number of accounts that run under the SQL Server Trusted Launchpad service.

- To configure a resource pool for managing external resources, see [Create an external resource pool](#).
- To change the amount of memory reserved for the database, see [Server memory configuration options](#).
- To change the number of R accounts that can be started by SQL Server Trusted Launchpad, see [Modify the user account pool for machine learning](#).

If you are using Standard Edition and do not have Resource Governor, you can use Dynamic Management Views (DMVs) and Extended Events, as well as Windows event monitoring, to help manage the server resources that are used by R. For more information, see [Monitoring and managing R Services](#).

Install additional R packages

The R solutions you create for SQL Server can call basic R functions, functions from the proprietary packages installed with SQL Server, and third-party R packages compatible with the version of open-source R installed by SQL Server.

Packages that you want to use from SQL Server must be installed in the default library that is used by the instance. If you have a separate installation of R on the computer, or if you installed packages to user libraries, you won't be able to use those packages from T-SQL.

The process for installing and managing R packages is different in SQL Server 2016 and SQL Server 2017. In SQL Server 2016, a database administrator must install R packages that users need. In SQL Server 2017, you can set up user groups to share packages on a per-database level, or configure database roles to enable users to install their own packages. For more information, see [Install new R packages](#).

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Install SQL Server 2016 R Server (Standalone)

7/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to use SQL Server 2016 setup to install the standalone version of **SQL Server 2016 R Server**.

Pre-install checklist

SQL Server 2016 is required. If you have SQL Server 2017, please install [SQL Server 2017 Machine Learning Server \(Standalone\)](#) instead.

If you installed any previous version of the Revolution Analytics tools or packages, you must uninstall them first.

Get the installation media

The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express Editions** are licensed for production use. For Enterprise and Standard Editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- **Free editions** are available at [SQL Server Downloads](#).

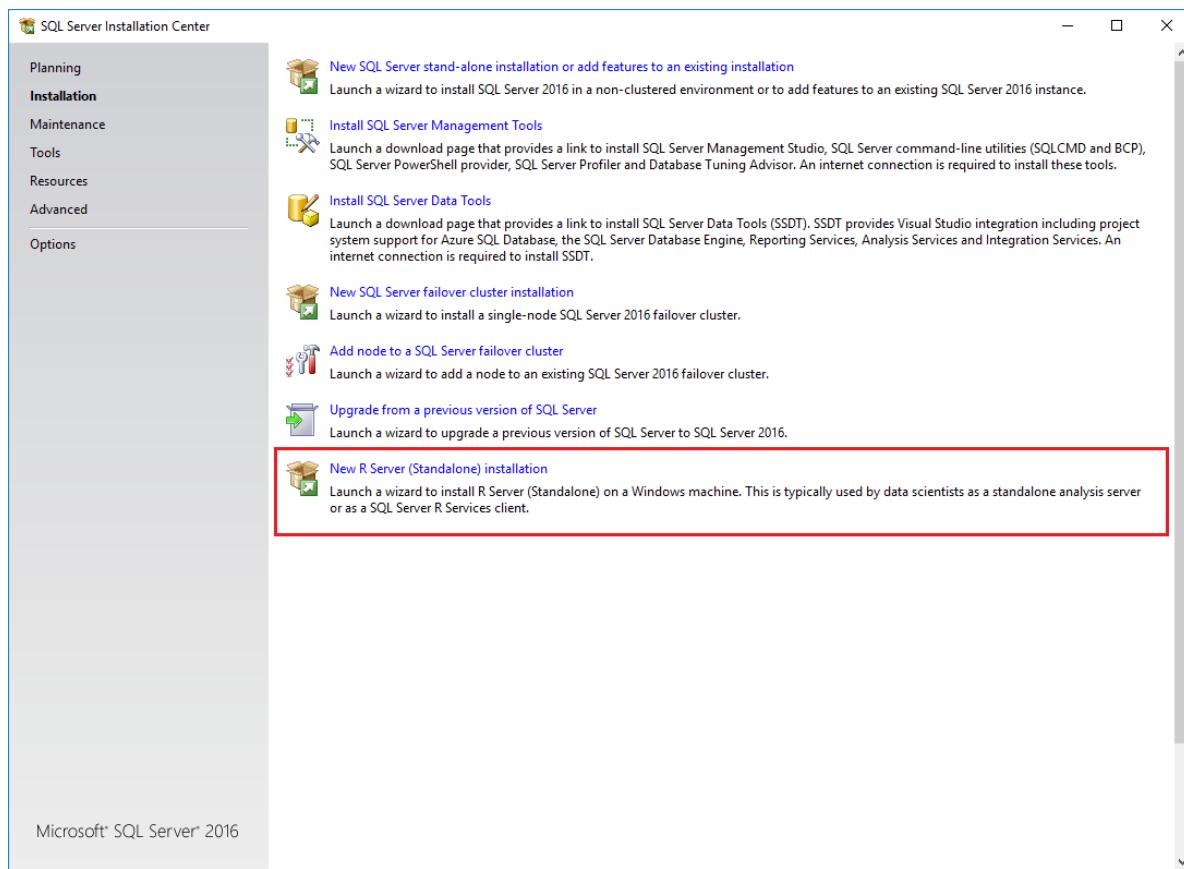
Install patch requirement

Microsoft has identified a problem with the specific version of Microsoft VC++ 2013 Runtime binaries that are installed as a prerequisite by SQL Server. If this update to the VC runtime binaries is not installed, SQL Server may experience stability issues in certain scenarios. Before you install SQL Server follow the instructions at [SQL Server Release Notes](#) to see if your computer requires a patch for the VC runtime binaries.

Run Setup

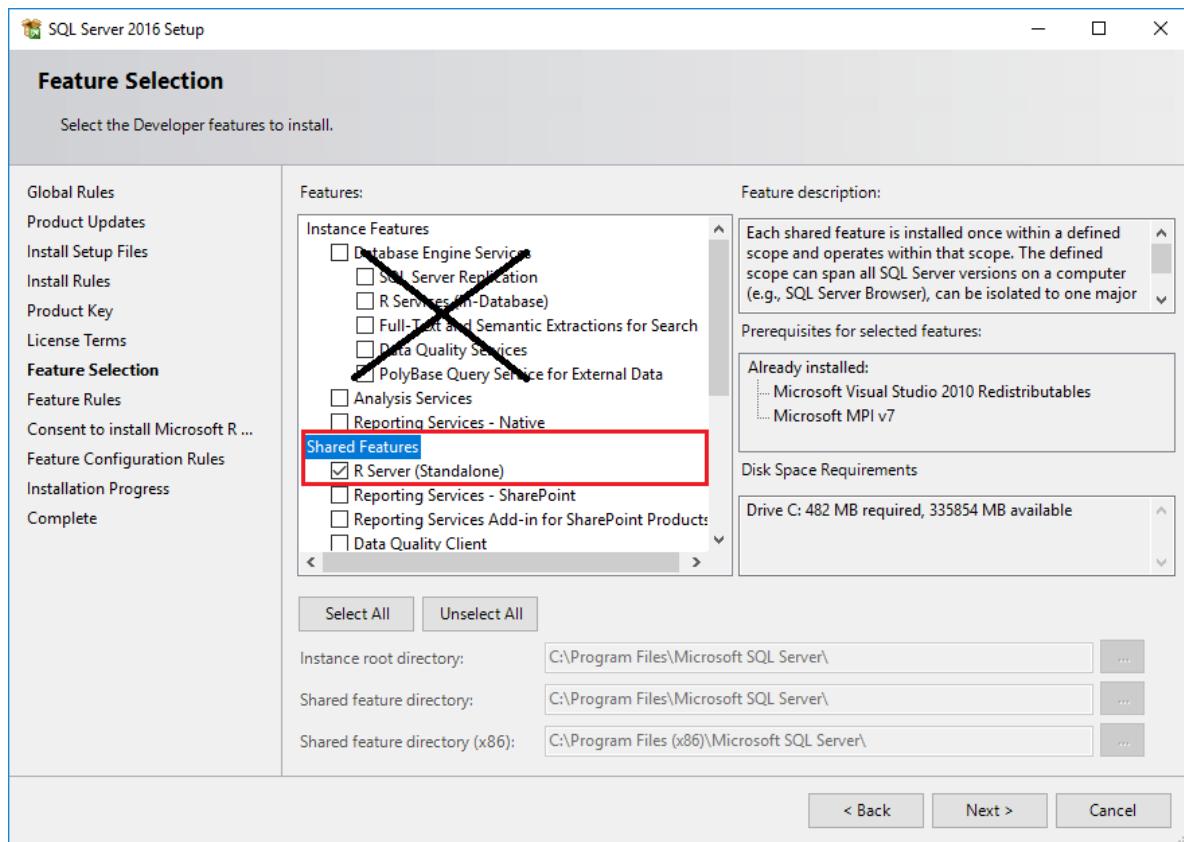
For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

1. Start the setup wizard for SQL Server 2016. We recommend that you install Service Pack 1 or later.
2. On the **Installation** tab, click **New R Server (Standalone) installation**.



3. On the **Feature Selection** page, the following option should be already selected:

R Server (Standalone)



All other options can be ignored.

NOTE

Avoid installing the **Shared Features** if you are running setup on a computer where R Services has already been installed for SQL Server in-database analytics. This creates duplicate libraries.

Whereas R scripts running in SQL Server are managed by SQL Server so as not to conflict with memory used by other database engine services, the standalone R Server has no such constraints, and can interfere with other database operations.

We generally recommend that you install R Server (Standalone) on a separate computer from SQL Server R Services (In-Database).

- Accept the license terms for downloading and installing Microsoft R Open. When the **Accept** button becomes unavailable, you can click **Next**.

Installation of these components, and any prerequisites they might require, might take a while.

- On the **Ready to Install** page, verify your selections, and click **Install**.

Default installation folders

When you install R Server using SQL Server setup, the R libraries are installed in a folder associated with the SQL Server version that you used for setup. In this folder, you will also find sample data, documentation for the R base packages, and documentation of the R tools and runtime.

However, if you installed Microsoft R Server using the separate Windows installer (not SQL Setup), or if you upgrade using the separate Windows installer, the R libraries are installed in a different folder.

Although we recommend against it, if you also installed an instance of SQL Server with R Services (In-Database) on the same computer, a second copy of R libraries and tools are installed in a different folder.

The following table lists the paths for each installation.

VERSION	INSTALLATION METHOD	DEFAULT FOLDER
R Server (Standalone)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\130\R_SERVER
R Server (Standalone)	Windows standalone installer	C:\Program Files\Microsoft\R Server\R_SERVER
Machine Learning Server (Standalone)	SQL Server 2017 setup wizard, with R language option	C:\Program Files\Microsoft SQL Server\140\R_SERVER
Machine Learning Server (Standalone)	SQL Server 2017 setup wizard, with Python language option	C:\Program Files\Microsoft SQL Server\140\PYTHON_SERVER
Machine Learning Server (Standalone)	Windows standalone installer	C:\Program Files\Microsoft\R Server\R_SERVER
R Services (In-Database)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\MSSQL13.<instance_name>\R_SERVICES
Machine Learning Services (In-Database)	SQL Server 2017 setup wizard, with R language option	C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\R_SERVICES

VERSION	INSTALLATION METHOD	DEFAULT FOLDER
Machine Learning Services (In-Database)	SQL Server 2017 setup wizard, with Python language option	C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\PYTHON_SERVICES

Development tools

A development IDE is not installed as part of setup. Additional tools are not required, as all the standard tools are included that would be provided with a distribution of R or Python.

We recommend that you try the new release of R Tools for Visual Studio or [Python for Visual Studio](#). Visual Studio supports both R and Python, as well as database development tools, connectivity with SQL Server, and BI tools. However, you can use any preferred development environment, including RStudio.

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#).
- [Tutorial: In-database analytics for R developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Install pre-trained machine learning models on SQL Server

7/23/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article explains how to use Powershell to add free pre-trained machine learning models for *sentiment analysis* and *image featurization* to a SQL Server database engine instance having R or Python integration. The pre-trained models are built by Microsoft and ready-to-use, added to a database engine instance as a post-install task. For more information about these models, see the [Resources](#) section of this article.

Once installed, the pre-trained models are considered an implementation detail that power specific functions in the MicrosoftML (R) and microsoftml (Python) libraries. You should not (and cannot) view, customize, or retrain the models, nor can you treat them as an independent resource in custom code or paired other functions.

To use the pretrained models, call the functions listed in the following table.

R FUNCTION (MICROSOFTML)	PYTHON FUNCTION (MICROSOFTML)	USAGE
<code>getSentiment</code>	<code>get_sentiment</code>	Generates positive-negative sentiment score over text inputs. Learn more .
<code>featurizeImage</code>	<code>featurize_image</code>	Extracts text information from image file inputs. Learn more .

Prerequisites

Machine learning algorithms are computationally intensive. We recommend 16 GB RAM for low-to-moderate workloads, including completion of the tutorial walkthroughs using all of the sample data.

You must have administrator rights on the computer and SQL Server to add pre-trained models.

External scripts must be enabled and SQL Server LaunchPad service must be running. Installation instructions provide the steps for enabling and verifying these capabilities.

[MicrosoftML R package](#) or [microsoftml Python package](#) contain the pre-trained models.

- [SQL Server 2017 Machine Learning Services](#) includes both language versions of the machine learning library, so this prerequisite is met with no further action on your part. Because the libraries are present, you can use the PowerShell script described in this article to add the pre-trained models to these libraries.
- [SQL Server 2016 R Services](#), which is R only, does not include [MicrosoftML package](#) out of the box. To add MicrosoftML, you must do a [component upgrade](#). One advantage of the component upgrade is that you can simultaneously add the pre-trained models, which makes running the PowerShell script unnecessary. However, if you already upgraded but missed adding the pre-trained models the first time around, you can run the PowerShell script as described in this article. It works for both versions of SQL Server. Before you do, confirm that the MicrosoftML library exists at C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library.

Check whether pre-trained models are installed

The install paths for R and Python models are as follows:

- For R:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\MicrosoftML\mxLibs\x64
```

- For Python:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\Lib\site-packages\microsoftml\mxLibs
```

Model file names are listed below:

- AlexNet_Updated.model
- ImageNet1K_mean.xml
- pretrained.model
- ResNet_101_Updated.model
- ResNet_18_Updated.model
- ResNet_50_Updated.model

If the models are already installed, skip ahead to the [validation step](#) to confirm availability.

Download the installation script

Click <https://aka.ms/mlm4sql> to download the file **Install-MLModels.ps1**.

Execute with elevated privileges

1. Start PowerShell. On the task bar, right-click the PowerShell program icon and select **Run as administrator**.
2. Enter a fully-qualified path to the installation script file and include the instance name. Assuming the Downloads folder and a default instance, the command might look like this:

```
PS C:\WINDOWS\system32> C:\Users\<user-name>\Downloads\Install-MLModels.ps1 MSSQLSERVER
```

Output

On an internet-connected SQL Server 2017 Machine Learning default instance with R and Python, you should see messages similar to the following.

```
MSSQL14.MSSQLSERVER
Verifying R models [9.2.0.24]
Downloading R models [C:\Users\<user-name>\AppData\Local\Temp]
Installing R models [C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\]
Verifying Python models [9.2.0.24]
Installing Python models [C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\]
PS C:\WINDOWS\system32>
```

Verify installation

First, check for the new files in the [mxlibs folder](#). Next, run demo code to confirm the models are installed and functional.

R verification steps

1. Start **RGUI.EXE** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64.
2. Paste in the following R script at the command prompt.

```

# Create the data
CustomerReviews <- data.frame(Review = c(
  "I really did not like the taste of it",
  "It was surprisingly quite good!",
  "I will never ever ever go to that place again!!"),
  stringsAsFactors = FALSE)

# Get the sentiment scores
sentimentScores <- rxFeaturize(data = CustomerReviews,
  mlTransforms = getSentiment(vars = list(SentimentScore = "Review")))

# Let's translate the score to something more meaningful
sentimentScores$PredictedRating <- ifelse(sentimentScores$SentimentScore > 0.6,
  "AWESOMENESS", "BLAH")

# Let's look at the results
sentimentScores

```

3. Press Enter to view the sentiment scores. Output should be as follows:

```

> sentimentScores
      Review SentimentScore
1 I really did not like the taste of it      0.4617899
2 It was surprisingly quite good!      0.9601924
3 I will never ever ever go to that place again!!      0.3103435
PredictedRating
1          BLAH
2      AWESOMENESS
3          BLAH

```

Python verification steps

1. Start **Python.exe** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES.
2. Paste in the following Python script at the command prompt

```

import numpy
import pandas
from microsoftml import rx_logistic_regression, rx_featurize, rx_predict, get_sentiment

# Create the data
customer_reviews = pandas.DataFrame(data=dict(review=[
  "I really did not like the taste of it",
  "It was surprisingly quite good!",
  "I will never ever ever go to that place again!!"]))

# Get the sentiment scores
sentiment_scores = rx_featurize(
  data=customer_reviews,
  ml_transforms=[get_sentiment(cols=dict(scores="review"))])

# Let's translate the score to something more meaningful
sentiment_scores["eval"] = sentiment_scores.scores.apply(
  lambda score: "AWESOMENESS" if score > 0.6 else "BLAH")
print(sentiment_scores)

```

3. Press Enter to print the scores. Output should be as follows:

```
>>> print(sentiment_scores)
           review      scores      eval
0       I really did not like the taste of it  0.461790    BLAH
1       It was surprisingly quite good!  0.960192  AWESOMENESS
2  I will never ever ever go to that place again!!  0.310344    BLAH
>>>
```

NOTE

If demo scripts fail, check the file location first. On systems having multiple instances of SQL Server, or for instances that run side-by-side with standalone versions, it's possible for the installation script to mis-read the environment and place the files in the wrong location. Usually, manually copying the files to the correct mxlib folder fixes the problem.

Examples using pre-trained models

The following links include walkthroughs and example code invoking the pretrained models.

- [Sentiment analysis with Python in SQL Server Machine Learning Services](#)
- [Image featurization with a pre-trained deep neural network model](#)

The pre-trained model for images supports featurization of images that you supply. To use the model, you call the **featurizeImage** transform. The image is loaded, resized, and featurized by the trained model. The output of the DNN featurizer is then used to train a linear model for image classification. To use this model, all images must be resized to meet the requirements of the trained model. For example, if you use an AlexNet model, the image should be resized to 227 x 227 px.

- [Code sample: Sentiment Analysis using Text Featurizer](#)

Research and resources

Currently the models that are available are deep neural network (DNN) models for sentiment analysis and image classification. All pre-trained models were trained by using Microsoft's [Computation Network Toolkit](#), or **CNTK**.

The configuration of each network was based on the following reference implementations:

- ResNet-18
- ResNet-50
- ResNet-101
- AlexNet

For more information about the algorithms used in these deep learning models, and how they are implemented and trained using CNTK, see these articles:

- [Microsoft Researchers' Algorithm Sets ImageNet Challenge Milestone](#)
- [Microsoft Computational Network Toolkit offers most efficient distributed deep learning computational performance](#)

See also

- [SQL Server 2016 R Services](#)
- [SQL Server 2017 Machine Learning Services](#)
- [Upgrade R and Python components in SQL Server instances](#)
- [MicrosoftML package for R](#)

- [microsoftml package for Python](#)

Install SQL Server machine learning components from the command line

6/1/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides instructions for installing SQL Server machine learning components from a command line:

- [New In-Database instance](#)
- [Add to an existing database engine instance](#)
- [Silent install](#)
- [New standalone server](#)

You can specify silent, basic, or full interaction with the Setup user interface. This article supplements [Install SQL Server from the Command Prompt](#), covering the parameters unique to R and Python machine learning components.

Pre-install checklist

- Run commands from an elevated command prompt.
- A database engine instance is required for in-database installations. You cannot install just R or Python features, although you can [add them incrementally to an existing instance](#). If you want just R and Python without the database engine, install the [standalone server](#).
- Do not install on a failover cluster. The security mechanism used for isolating R and Python processes is not compatible with a Windows Server failover cluster environment.
- Do not install on a domain controller. The Machine Learning Services portion of setup will fail.
- Avoid installing standalone and in-database instances on the same computer. A standalone server will compete for the same resources, undermining the performance of both installations.

Command line arguments

The FEATURES argument is required, as are licensing term agreements.

When installing through the command prompt, SQL Server supports full quiet mode by using the /Q parameter, or Quiet Simple mode by using the /QS parameter. The /QS switch only shows progress, does not accept any input, and displays no error messages if encountered. The /QS parameter is only supported when /Action=install is specified.

ARGUMENTS	DESCRIPTION
/FEATURES = AdvancedAnalytics	Installs the in-database version: SQL Server 2017 Machine Learning Services (In-Database) or SQL Server 2016 R Services (In-Database).

ARGUMENTS	DESCRIPTION
/FEATURES = SQL_INST_MR	Applies to SQL Server 2017 only. Pair this with AdvancedAnalytics. Installs the (In-Database) R feature, including Microsoft R Open and the proprietary R packages. The SQL Server 2016 R Services feature is R-only, so there is no parameter for that release.
/FEATURES = SQL_INST_MPY	Applies to SQL Server 2017 only. Pair this with AdvancedAnalytics. Installs the (In-Database) Python feature, including Anaconda and the proprietary Python packages.
/FEATURES = SQL_SHARED_MR	Installs the R feature for the standalone version: SQL Server 2017 Machine Learning Server (Standalone) or SQL Server 2016 R Server (Standalone). A standalone server is a "shared feature" not bound to a database engine instance.
/FEATURES = SQL_SHARED_MPY	Applies to SQL Server 2017 only. Installs the Python feature for the standalone version: SQL Server 2017 Machine Learning Server (Standalone). A standalone server is a "shared feature" not bound to a database engine instance.
/IACCEPTOPENLICENSETERMS	Indicates you have accepted the license terms for using the open source R components.
/IACCEPPTYTHONLICENSETERMS	Indicates you have accepted the license terms for using the Python components.
/IACCEPTSQLSERVERLICENSETERMS	Indicates you have accepted the license terms for using SQL Server.
/MRCACHEDIRECTORY	For offline setup, sets the folder containing the R component CAB files.
/MPYCACHEDIRECTORY	For offline setup, sets the folder containing the Python component CAB files.

In-database instance installations

In-database analytics are available for database engine instances, required for adding the **AdvancedAnalytics** feature to your installation. You can install a database engine instance with advanced analytics, or [add it to an existing instance](#).

To view progress information without the interactive on-screen prompts, use the /qs argument.

IMPORTANT

After installation, two additional configuration steps remain. Integration is not complete until these tasks are performed. See [Post-installation tasks](#) for instructions.

SQL Server 2017: database engine, advanced analytics with Python and R

For a concurrent installation of the database engine instance, provide the instance name and an administrator (Windows) login. Include features for installing core and language components, as well as acceptance of all licensing terms.

```
Setup.exe /qs /ACTION=Install /FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<Windows-username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS /IACCEPPTYTHONLICENSETERMS
```

This is the same command, but with a SQL Server login on a database engine using mixed authentication.

```
Setup.exe /q /ACTION=Install /FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SECURITYMODE=SQL /SAPWD="%password%" /SQLSYSADMINACCOUNTS="<sql-username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS /IACCEPPTYTHONLICENSETERMS
```

This example is Python only, showing that you can add one language by omitting a feature.

```
Setup.exe /qs /ACTION=Install /FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPPTYTHONLICENSETERMS
```

SQL Server 2016: database engine and advanced analytics with R

This command is identical to SQL Server 2017, but without the Python elements, which are not available in SQL Server 2016 setup.

```
Setup.exe /qs /ACTION=Install /FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<Windows-username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS
```

Post-install configuration (required)

Applies to in-database installations only.

When setup is finished, you have a database engine instance with R and Python, the Microsoft R and Python packages, Microsoft R Open, Anaconda, tools, samples, and scripts that are part of the distribution.

Two more tasks are required to complete the installation:

1. Restart the database engine service.
2. Enable external scripts before you can use the feature. Follow the instructions in [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#) as your next step.

For SQL Server 2016, use this article instead [Install SQL Server 2016 R Services \(In-Database\)](#).

Add advanced analytics to an existing database engine instance

When adding in-database advanced analytics to an existing database engine instance, provide the instance name. For example, if you previously installed a SQL Server 2017 database engine and Python, you could use this command to add R.

```
Setup.exe /qs /ACTION=Install /FEATURES=SQL_INST_MR /INSTANCENAME=MSSQLSERVER  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS
```

Silent install

A silent installation suppresses the check for .cab file locations. For this reason, you must specify the location where .cab files are to be unpacked. You can use the temp directory for this.

```
Setup.exe /q /ACTION=Install /FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS=<username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS /IACCEPPTYTHONLICENSETERMS  
/MRCACHEDIRECTORY=%temp% /MPYCACHEDIRECTORY=%temp%
```

Standalone server installations

A standalone server is a "shared feature" not bound to a database engine instance. The following examples show valid syntax for both releases.

SQL Server 2017 supports Python and R on a standalone server:

```
Setup.exe /q /ACTION=Install /FEATURES=SQL_SHARED_MR,SQL_SHARED_MPY  
/IACCEPTOPENLICENSETERMS /IACCEPPTYTHONLICENSETERMS /IACCEPTSQLSERVERLICENSETERMS
```

SQL Server 2016 is R-only:

```
Setup.exe /q /ACTION=Install /FEATURES=SQL_SHARED_MR  
/IACCEPTOPENLICENSETERMS /IACCEPTSQLSERVERLICENSETERMS
```

When setup is finished, you have a server, Microsoft packages, open-source distributions of R and Python, tools, samples, and scripts that are part of the distribution.

To open an R console window, go to \Program files\Microsoft SQL Server\140 (or 130)\R_SERVER\bin\x64 and double-click **RGui.exe**. New to R? Try this tutorial: [Basic R commands and RevoScaleR functions: 25 common examples](#).

To open a Python command, go to \Program files\Microsoft SQL Server\140\PYTHON_SERVER\bin\x64 and double-click **python.exe**.

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Tutorial: Run Python in T-SQL](#)
- [Tutorial: In-database analytics for Python developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Install SQL Server machine learning components without internet access

6/6/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

By default, installers connect to Microsoft download sites to get required and updated components for machine learning on SQL Server. If firewall constraints prevent the installer from reaching these sites, you can use an internet-connected device to download files, transfer files to an offline server, and then run setup.

Get the installation media

The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express Editions** are licensed for production use. For Enterprise and Standard Editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- **Free editions** are available at [SQL Server Downloads](#).

NOTE

For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

Install patch requirement

Microsoft has identified a problem with the specific version of Microsoft VC++ 2013 Runtime binaries that are installed as a prerequisite by SQL Server. If this update to the VC runtime binaries is not installed, SQL Server may experience stability issues in certain scenarios. Before you install SQL Server follow the instructions at [SQL Server Release Notes](#) to see if your computer requires a patch for the VC runtime binaries.

Download .cab files

On an internet-connected server, download the .cab files required for an offline installation. The setup program uses the .cab files to install supplemental features.

RELEASE	DOWNLOAD LINK
SQL Server 2017 initial release	
Microsoft R Open	SRO_3.3.3.24_1033.cab
Microsoft R Server	SRS_9.2.0.24_1033.cab
Microsoft Python Open	SPO_9.2.0.24_1033.cab
Microsoft Python Server	SPS_9.2.0.24_1033.cab

RELEASE	DOWNLOAD LINK
SQL Server 2017 CU1	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_9.2.0.100_1033.cab
Microsoft Python Open	no change; use previous
Microsoft Python Server	SPS_9.2.0.100_1033.cab
SQL Server 2017 CU2	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
Microsoft Python Open	no change; use previous
Microsoft Python Server	no change; use previous
SQL Server 2017 CU3	
Microsoft R Open	SRO_3.3.3.300_1033.cab
Microsoft R Server	SRS_9.2.0.300_1033.cab
Microsoft Python Open	no change; use previous
Microsoft Python Server	SPS_9.2.0.300_1033.cab
SQL Server 2017 CU4	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_9.2.0.400_1033.cab
Microsoft Python Open	no change; use previous
Microsoft Python Server	SPS_9.2.0.400_1033.cab
SQL Server 2017 CU5	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_9.2.0.500_1033.cab
Microsoft Python Open	no change; use previous
Microsoft Python Server	SPS_9.2.0.500_1033.cab

RELEASE	DOWNLOAD LINK
SQL Server 2017 CU6	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_9.2.0.600_1033.cab
Microsoft Python Open	no change; use previous
Microsoft Python Server	SPS_9.2.0.600_1033.cab
SQL Server 2017 CU7	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
Microsoft Python Open	no change; use previous
Microsoft Python Server	no change; use previous

Downloads for SQL Server 2016

IMPORTANT

When installing SQL Server 2016 SP1 CU4 or SP1 CU5 offline, download SRO_3.2.2.16000_1033.cab. If you downloaded SRO_3.2.2.13000_1033.cab from FWLINK 831785 as indicated in the setup dialog box, rename the file as SRO_3.2.2.16000_1033.cab before installing the Cumulative Update.

RELEASE	DOWNLOAD LINK
SQL Server 2016 RTM	
Microsoft R Open	SRO_3.2.2.803_1033.cab
Microsoft R Server	SRS_8.0.3.0_1033.cab
SQL Server 2016 CU 1	
Microsoft R Open	SRO_3.2.2.10000_1033.cab
Microsoft R Server	SRS_8.0.3.10000_1033.cab
SQL Server 2016 CU 2	
Microsoft R Open	SRO_3.2.2.12000_1033.cab
Microsoft R Server	SRS_8.0.3.12000_1033.cab
SQL Server 2016 CU 3	

RELEASE	DOWNLOAD LINK
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
SQL Server 2016 CU 4	
Microsoft R Open	SRO_3.2.2.13000_1033.cab
Microsoft R Server	SRS_8.0.3.13000_1033.cab
SQL Server 2016 CU 5	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
SQL Server 2016 CU 6	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_8.0.3.14000_1033.cab
SQL Server 2016 CU 7	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
SQL Server 2016 SP 1	
Microsoft R Open	SRO_3.2.2.15000_1033.cab
Microsoft R Server	SRS_8.0.3.15000_1033.cab
SQL Server 2016 SP 1 CU1	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous
SQL Server 2016 SP 1 CU2	
Microsoft R Open	SRO_3.2.2.16000_1033.cab
Microsoft R Server	SRS_8.0.3.16000_1033.cab
SQL Server 2016 SP 1 CU3	
Microsoft R Open	no change; use previous

RELEASE	DOWNLOAD LINK
Microsoft R Server	no change; use previous
SQL Server 2016 SP 1 CU4 and GDR	
Microsoft R Open	no change; use previous
Microsoft R Server	SRS_8.0.3.17000_1033.cab
SQL Server 2016 SP 1 CU5	
Microsoft R Open	no change; use previous
Microsoft R Server	no change; use previous

If you would like to view the source code for Microsoft R, it is available for download as an archive in .tar format:

[Download R Server installers](#)

Additional prerequisites

Depending on your environment, you might need to make local copies of installers for the following prerequisites.

COMPONENT	VERSION
Microsoft AS OLE DB Provider for SQL Server 2016	13.0.1601.5
Microsoft .NET Core	1.0.1
Microsoft MPI	7.1.12437.25
Microsoft Visual C++ 2013 Redistributable	12.0.30501.0
Microsoft Visual C++ 2015 Redistributable	14.0.23026.0

Transfer files

Transfer the zipped SQL Server installation media and the files you already downloaded to the computer on which you are installing setup.

Put the CAB files in a convenient folder such as **Downloads** or the setup user's temp folder:
C:\Users\AppData\Local\Temp.

Put the en_sql_server_2017.iso file in a convenient folder. Double-click **setup.exe** to begin installation.

Run Setup

When you run SQL Server setup on a computer disconnected from the internet, Setup adds an **Offline installation** page to the wizard so that you can specify the location of the .cab files you copied in the previous step.

1. Start the SQL Server setup wizard.
2. When the setup wizard displays the licensing page for open source R or Python components, click **Accept**. Acceptance of licensing terms allows you to proceed to the next step.
3. In the **Offline installation** page, in **Install Path**, specify the folder containing the .cab files you copied

earlier.

4. Continue following the on-screen prompts to complete the installation.

After installation is finished, restart the service and then configure the server to enable script execution as described in [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#) or [Install SQL Server 2016 R Services \(In-Database\)](#).

Slipstream upgrades for offline servers

Slipstream setup refers to the ability to apply a patch or update to a failed instance installation, to repair existing problems. The advantage of this method is that the SQL Server is updated at the same time that you perform setup, avoiding a separate restart later.

- If the server does not have Internet access, you must download the SQL Server installer, and then download matching versions of the R component installers **before** beginning the update process. The R components are not included by default with SQL Server.
- If you are adding these components to an existing installation, use the updated version of the SQL Server installer, and the corresponding updated version of the additional components. When you specify that the R feature is to be installed, the installer looks for the matching version of the installers for the machine learning components.

Get help

Need help with installation or upgrade? For answers to common questions and known issues, see the following article:

- [Upgrade and installation FAQ - Machine Learning Services](#)

To check the installation status of the instance and fix common issues, try these custom reports.

- [Custom reports for SQL Server R Services](#)

This article by the R Services Support team demonstrates how to perform an unattended install or upgrade of R services in SQL Server 2016: [Deploying R Services on Computers without Internet Access](#).

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Tutorial: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Tutorial: Run Python in T-SQL](#)
- [Tutorial: In-database analytics for Python developers](#)

To view examples of machine learning that are based on real-world scenarios, see [Machine learning tutorials](#).

Upgrade machine learning (R and Python) components in SQL Server instances

7/20/2018 • 15 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

R and Python integration in SQL Server includes open-source and Microsoft-proprietary packages. Under standard SQL Server servicing, R and Python packages are updated according to the SQL Server release cycle, with bug fixes to existing packages at the current version.

Most data scientists are accustomed to working with newer packages as they become available. For both SQL Server 2017 Machine Learning Services (In-Database) and SQL Server 2016 R Services (In-Database), you can get newer versions of R and Python by changing the *binding* from SQL Server servicing to [Microsoft Machine Learning Server](#) and the [Modern Lifecycle Support policy](#).

Binding does not change the fundamentals of your installation: R and Python integration is still part of a database engine instance, licensing is unchanged (no additional costs associated with binding), and SQL Server support policies still hold for the database engine. But rebinding does change how R and Python packages are serviced. The rest of this article explains the binding mechanism and how it works for each version of SQL Server.

NOTE

Binding applies to (In-Database) instances only. Binding is not relevant for a (Standalone) installation.

SQL Server 2017 binding considerations

For SQL Server 2017 Machine Learning Services, you would consider binding only when Microsoft Machine Learning Server begins to offer additional packages or newer versions over what you already have.

SQL Server 2016 binding considerations

For SQL Server 2016 R Services customers, binding provides updated R packages, new packages not part of the original installation ([MicrosoftML](#)), and [pretrained models](#), all of which can further be refreshed at each new major and minor release of Microsoft Machine Learning Server. Binding does not give you Python support, which is a SQL Server 2017 feature.

Version map

The following table is a version map, showing package versions across release vehicles so that you can ascertain potential upgrade paths when you bind to Microsoft Machine Learning Server (previously known as R Server, before the addition of Python support starting in MLS 9.2.1).

Notice that binding does not guarantee the very latest version of R or Anaconda. When you bind to Microsoft Machine Learning Server (MLS), you get the R or Python version installed through Setup, which may not be the latest version available on the web.

[SQL Server 2016 R Services](#)

COMPONENT	INITIAL RELEASE	R SERVER 9.0.1	R SERVER 9.1	MLS 9.2.1	MLS 9.3
Microsoft R Open (MRO) over R	R 3.2.2	R 3.3.2	R 3.3.3	R 3.4.1	R 3.4.3
RevoScaleR	8.0.3	9.0.1	9.1	9.2.1	9.3
MicrosoftML	n.a.	9.0.1	9.1	9.2.1	9.3
pretrained models	n.a.	9.0.1	9.1	9.2.1	9.3
sqlrutils	n.a.	1.0	1.0	1.0	1.0
olapR	n.a.	1.0	1.0	1.0	1.0

SQL Server 2017 Machine Learning Services

COMPONENT	INITIAL RELEASE	MLS 9.3			
Microsoft R Open (MRO) over R	R 3.3.3	R 3.4.3			
RevoScaleR	9.2	9.3			
MicrosoftML	9.2	9.3			
sqlrutils	1.0	1.0			
olapR	1.0	1.0			
Anaconda 4.2 over Python 3.5	4.2/3.5.2	4.2/3.5.2			
revoscalepy	9.2	9.3			
microsoftml	9.2	9.3			
pretrained models	9.2	9.3			

How component upgrade works

Component upgrade occurs when you *bind* a SQL Server 2016 R Services instance (or a SQL Server 2017 Machine Learning Services instance) to [Microsoft Machine Learning Server](#). This process basically overwrites the contents of C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES as installed by SQL Server Setup with the contents of C:\Program Files\Microsoft\ML Server\R_SERVER.

Microsoft Machine Learning Server is an on-premises server product separate from SQL Server, but with the same interpreters and packages. Binding swaps out the SQL Server service update mechanism so that you can use the R and Python packages shipping with Microsoft Machine Learning Server, which are often newer than those installed by SQL Server. Switching support policies is an attractive option for data science teams who

require newer generation R and Python modules for their solutions.

Binding is executed by the [MLS installer](#). The installer updates specific R and Python packages, but does not replace your SQL Server in-database instance with a standalone, disconnected server install.

- Without binding, R and Python packages are patched for bug fixes when you install a SQL Server service pack or cumulative update (CU).
- With binding, newer package versions can be applied to your instance, independently of the CU release schedule, under the [Modern Lifecycle Policy](#) and releases of Microsoft Machine Learning Server. The Modern Lifecycle support policy offers more frequent updates over a shorter, one-year lifespan. Post-binding, you would continue to use the MLS installer for future updates of R and Python as they become available in Microsoft Machine Learning Server.

Binding applies to R and Python features only. Namely, open-source packages for R and Python features (Microsoft R Open, Anaconda), and the proprietary packages RevoScaleR, revoscalepy, and so forth. Binding does not change the support model for the database engine instance and doesn't change the version of SQL Server.

Binding is reversible. You can revert to SQL Server servicing by [unbinding the instance](#) and repairing your SQL Server database engine instance.

Summed up, steps for binding are as follows:

- Start with an existing, configured installation of SQL Server 2016 R Services (or SQL Server 2017 Machine Learning Services).
- Determine which version of Microsoft Machine Learning Server has the upgraded components you want to use.
- Download and run Setup for that version. Setup detects the existing instance, adds a binding option, and returns a list of compatible instances.
- Choose the instance you want to bind and then finish setup to execute the binding.

In terms of user experience, the technology and how you work with it is unchanged. The only difference is the presence of newer-versioned packages and possibly additional packages not originally available through SQL Server (such as MicrosoftML for SQL Server 2016 R Services customers).

Bind to MLS using Setup

Microsoft Machine Learning Setup detects the existing features and SQL Server version and invokes a utility called SqlBindR.exe to change the binding. Internally, SqlBindR is chained to Setup and used indirectly. Later, you can call SqlBindR directly from the command line to exercise specific options.

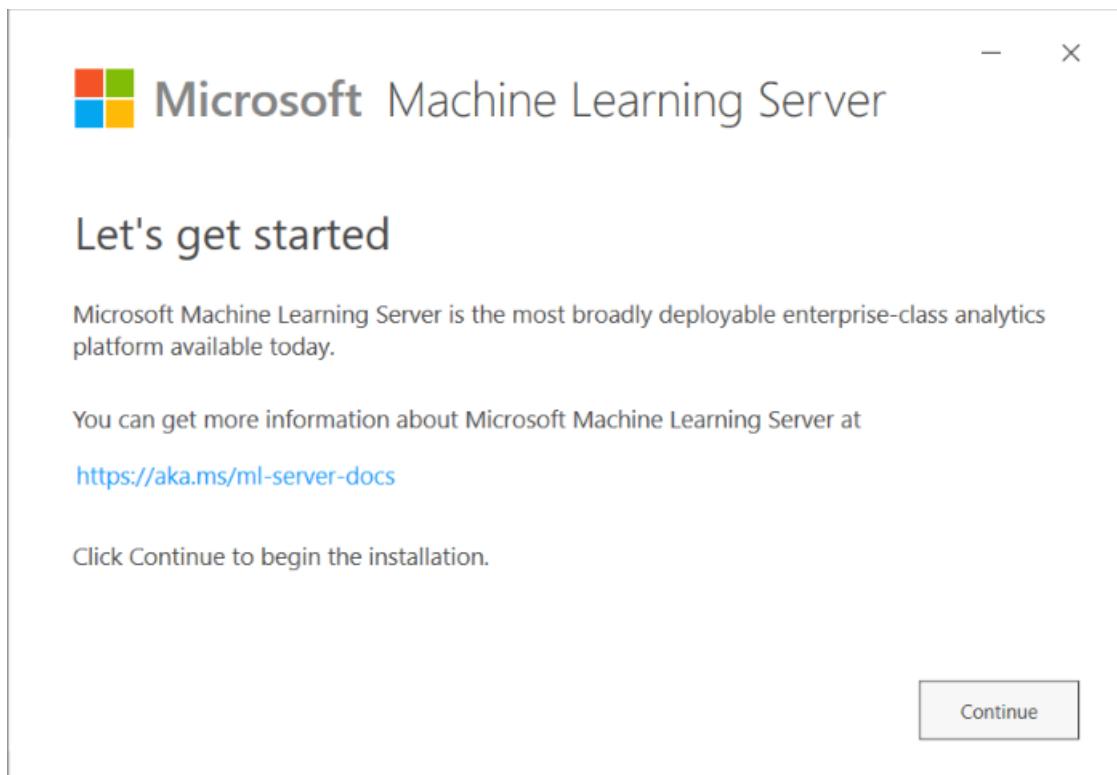
- In SSMS, run `SELECT @@version` to verify the server meets minimum build requirements.

For SQL Server 2016 R Services, the minimum is [Service Pack 1](#) and [CU3](#).

- Check the version of R base and RevoScaleR packages to confirm the existing versions are lower than what you plan to replace them with. For SQL Server 2016 R Services, R Base package is 3.2.2 and RevoScaleR is 8.0.3.

```
EXECUTE sp_execute_external_script
@language=N'R'
,@script = N'str(ResultSet);
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
ResultSet <- data.frame(Name, Version);
, @input_data_1 = N''
WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max) ))
```

3. Close down SSMS and any other tools having an open connection to SQL Server. Binding overwrites program files. If SQL Server has open sessions, binding will fail with bind error code 6.
4. Download Microsoft Machine Learning Server onto the computer that has the instance you want to upgrade. We recommend the [latest version](#).
5. Unzip the folder and start ServerSetup.exe, located under MLSWIN93.



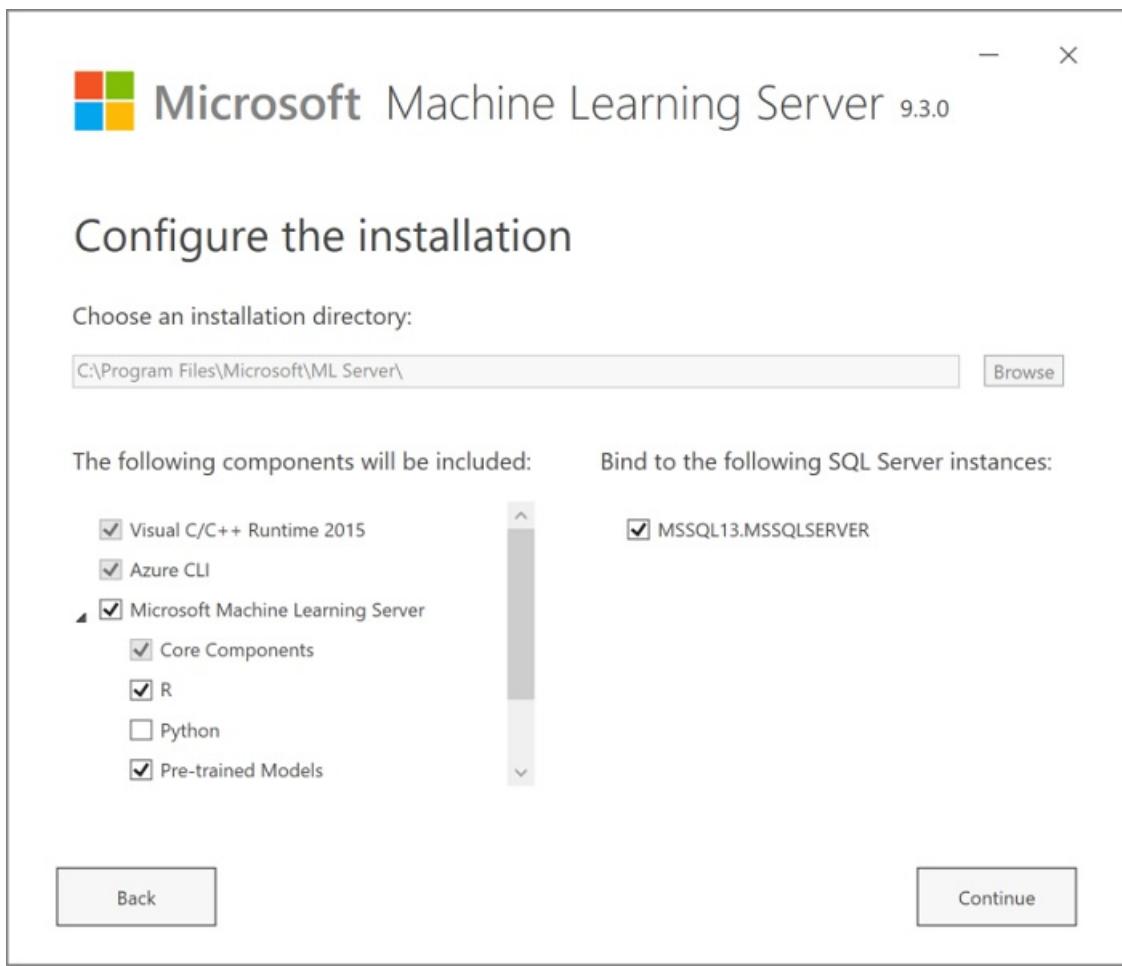
6. On **Configure the installation**, confirm the components to upgrade, and review the list of compatible instances.

This step is very important.

On the left, choose every feature that you want to keep or upgrade. You cannot upgrade some features and not others. An empty checkbox removes that feature, assuming it is currently installed. In the screenshot, given an instance of SQL Server 2016 R Services (MSSQL13), R and the R version of the pre-trained models are selected. This configuration is valid because SQL Server 2016 supports R but not Python.

If you are upgrading components on SQL Server 2016 R Services, do not select the Python feature. You cannot add Python to SQL Server 2016 R Services.

On the right, select the checkbox next to the instance name. If no instances are listed, you have an incompatible combination. If you do not select an instance, a new standalone installation of Machine Learning Server is created, and the SQL Server libraries are unchanged. If you can't select an instance, it might not be at [SP1 CU3](#).



7. On the **License agreement** page, select **I accept these terms** to accept the licensing terms for Machine Learning Server.
8. On successive pages, provide consent to additional licensing conditions for any open-source components you selected, such as Microsoft R Open or the Python Anaconda distribution.
9. On the **Almost there** page, make a note of the installation folder. The default folder is \Program Files\Microsoft\ML Server.

If you want to change the installation folder, click **Advanced** to return to the first page of the wizard. However, you must repeat all previous selections.

During the installation process, any R or Python libraries used by SQL Server are replaced and Launchpad is updated to use the newer components. As a result, if the instance previously used libraries in the default R_SERVICES folder, after upgrade these libraries are removed and the properties for the Launchpad service are changed to use the libraries in the new location.

Binding affects the contents of these folders: C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library is replaced with the contents of C:\Program Files\Microsoft\ML Server\R_SERVER. The second folder and its contents are created by Microsoft Machine Learning Server Setup.

If upgrade fails, check [SqlBindR error codes](#) for more information.

Confirm binding

Recheck the version of R and RevoScaleR to confirm you have newer versions. Use the R console distributed with the R packages in your database engine instance to get package information:

```

EXECUTE sp_execute_external_script
@language=N'R'
,@script = N'str(OutputDataSet);
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
OutputDataSet <- data.frame(Name, Version)';
, @input_data_1 = N''
WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max) ))

```

For SQL Server 2016 R Services bound to Machine Learning Server 9.3, R Base package should be 3.4.1, RevoScaleR should be 9.3, and you should also have MicrosoftML 9.3.

If you added the pre-trained models, the models are embedded in the MicrosoftML library and you can call them through MicrosoftML functions. For more information, see [R samples for MicrosoftML](#).

Offline binding (no internet access)

For systems with no internet connectivity, you can download the installer and .cab files to an internet-connected machine, and then transfer files to the isolated server.

The installer (ServerSetup.exe) includes the Microsoft packages (RevoScaleR, MicrosoftML, olapR, sqlRUtils). The .cab files provide other core components. For example, the "SRO" cab provides R Open, Microsoft's distribution of open-source R.

The following instructions explain how to place the files for an offline installation.

1. Download the MLS Installer. It downloads as a single zipped file. We recommend the [latest version](#), but you can also install [earlier versions](#).
2. Download .cab files. The following links are for the 9.3 release. If you require earlier versions, additional links can be found in [R Server 9.1](#). Recall that Python/Anaconda can only be added to a SQL Server 2017 Machine Learning Services instance. Pre-trained models exist for both R and Python; the .cab provides models in the languages you are using.

FEATURE	DOWNLOAD
R	SRO_3.4.3.0_1033.cab
Python	SPO_9.3.0.0_1033.cab
Pre-trained models	MLM_9.3.0.0_1033.cab

3. Transfer .zip and .cab files to the target server.
4. On the server, type `%temp%` in the Run command to get the physical location of the temp directory. The physical path varies by machine, but it is usually `C:\Users\<your-user-name>\AppData\Local\Temp`.
5. Place the .cab files in the %temp% folder.
6. Unzip the Installer.
7. Run ServerSetup.exe and follow the on-screen prompts to complete the installation.

Command line operations

After you run Microsoft Machine Learning Server, a command-line utility called SqlBindR.exe becomes available that you can use for further binding operations. For example, should you decide to reverse a binding, you could

either rerun Setup or use the command-line utility. Additionally, you can use this tool to check for instance compatibility and availability.

TIP

Can't find SqlBindR? You probably have not run Setup. SqlBindR is available only after running Machine Learning Server Setup.

1. Open a command prompt as administrator and navigate to the folder containing `sqlbindr.exe`. The default location is `C:\Program Files\Microsoft\MLServer\Setup`
2. Type the following command to view a list of available instances: `SqlBindR.exe /list`

Make a note of the full instance name as listed. For example, the instance name might be `MSSQL14.MSSQLSERVER` for a default instance, or something like `SERVERTNAME.MYNAMEDINSTANCE`.

3. Run the **SqlBindR.exe** command with the `/bind` argument, and specify the name of the instance to upgrade, using the instance name that was returned in the previous step.

For example, to upgrade the default instance, type: `SqlBindR.exe /bind MSSQL14.MSSQLSERVER`

4. When the upgrade has completed, restart the Launchpad service associated with any instance that has been modified.

Revert or unbind an instance

You can restore a bound instance to an initial installation of the R and Python components, established by SQL Server Setup. There are three parts to reverting back to the SQL Server servicing.

- [Step 1: Unbind from Microsoft Machine Learning Server](#)
- [Step 2: Restore the instance to original status](#)
- [Step 3: Reinstall any packages you added to the installation](#)

Step 1: Unbind

You have two options for rolling back the binding: re-rerun setup or use `SqlBindR` command line utility.

Unbind using Setup

1. Locate the installer for Machine Learning Server. If you have removed the installer, you might need to download it again, or copy it from another computer.
2. Be sure to run the installer on the computer that has the instance you want to unbind.
3. The installer identifies local instances that are candidates for unbinding.
4. Deselect the check box next to the instance that you want to revert to the original configuration.
5. Accept the licensing agreement. You must indicate your acceptance of licensing terms even when installing.
6. Click **Finish**. The process takes a while.

Unbind using the command line

1. Open a command prompt and navigate to the folder that contains **sqlbindr.exe**, as described in the previous section.
2. Run the **SqlBindR.exe** command with the `/unbind` argument, and specify the instance.

For example, the following command reverts the default instance:

```
SqlBindR.exe /unbind MSSQL14.MSSQLSERVER
```

Step 2: Repair the SQL Server instance

Run SQL Server Setup to repair the database engine instance having the R and Python features. Existing updates are preserved, but if you missed any SQL Server servicing updates to R and Python packages, this step applies those patches.

Alternatively, this is more work, but you could also fully uninstall and reinstall the database engine instance, and then apply all service updates.

Step 3: Add any third-party packages

You might have added other open-source or third-party packages to your package library. Since reversing the binding switches the location of the default package library, you must reinstall the packages to the library that R and Python are now using. For more information, see [Default packages](#), [Install new R packages](#), and [Install new Python packages](#).

SqlBindR.exe command syntax

Usage

```
sqlbindr [/list] [/bind <SQL_instance_ID>] [/unbind <SQL_instance_ID>]
```

Parameters

NAME	DESCRIPTION
<i>list</i>	Displays a list of all SQL database instance IDs on the current computer
<i>bind</i>	Upgrades the specified SQL database instance to the latest version of R Server and ensures the instance automatically gets future upgrades of R Server
<i>unbind</i>	Uninstalls the latest version of R Server from the specified SQL database instance and prevents future R Server upgrades from affecting the instance

Binding errors

MLS Installer and SqlBindR both return the following error codes and messages.

ERROR CODE	MESSAGE	DETAILS
Bind error 0	Ok (success)	Binding passed with no errors.
Bind error 1	Invalid arguments	Syntax error.
Bind error 2	Invalid action	Syntax error.
Bind error 3	Invalid instance	An instance exists, but is not valid for binding.
Bind error 4	Not bindable	
Bind error 5	Already bound	You ran the <i>bind</i> command, but the specified instance is already bound.

ERROR CODE	MESSAGE	DETAILS
Bind error 6	Bind failed	An error occurred while unbinding the instance. This error can occur if you run the MLS installer without selecting any features. Binding requires that you select both an MSSQL instance and R and Python, assuming the instance is SQL Server 2017. This error also occurs if SqlBindR could not write to the Program Files folder. Open sessions or handles to SQL Server will cause this error to occur. If you get this error, reboot the computer and redo the binding steps before starting any new sessions.
Bind error 7	Not bound	The database engine instance has R Services or SQL Server Machine Learning Services. The instance is not bound to Microsoft Machine Learning Server.
Bind error 8	Unbind failed	An error occurred while unbinding the instance.
Bind error 9	No instances found	No database engine instances were found on this computer.

Known issues

This section lists known issues specific to use of the SqlBindR.exe utility, or to upgrades of Machine Learning Server that might affect SQL Server instances.

Restoring packages that were previously installed

If you upgraded to Microsoft R Server 9.0.1, the version of SqlBindR.exe for that version failed to restore the original packages or R components completely, requiring that the user run SQL Server repair on the instance, apply all service releases, and then restart the instance.

Later version of SqlBindR automatically restore the original R features, eliminating the need for reinstallation of R components or re-patch the server. However, you must install any R package updates that might have been added after the initial installation.

If you have used the package management roles to install and share package, this task is much easier: you can use R commands to synchronize installed packages to the file system using records in the database, and vice versa. For more information, see [R package management for SQL Server](#).

Problems with multiple upgrades from SQL Server

If you have previously upgraded an instance of SQL Server 2016 R Services to 9.0.1, when you run the new installer for Microsoft R Server 9.1.0, it displays a list of all valid instances, and then by default selects previously bound instances. If you continue, the previously bound instances are unbound. As a result, the earlier 9.0.1 installation is removed, including any related packages, but the new version of Microsoft R Server (9.1.0) is not installed.

As a workaround, you can modify the existing R Server installation as follows:

1. In Control Panel, open **Add or Remove Programs**.

2. Locate Microsoft R Server, and click **Change/Modify**.
3. When the installer starts, select the instances you want to bind to 9.1.0.

Microsoft Machine Learning Server 9.2.1 and 9.3 do not have this issue.

Binding or unbinding leaves multiple temporary folders

Sometimes the binding and unbinding operations fail to clean up temporary folders. If you find folders with a name like this, you can remove it after installation is complete: R_SERVICES_

NOTE

Be sure to wait until installation is complete. It can take a long time to remove R libraries associated with one version and then add the new R libraries. When the operation completes, temporary folders are removed.

See also

- [Install Machine Learning Server for Windows \(Internet connected\)](#)
- [Install Machine Learning Server for Windows \(offline\)](#)
- [Known issues in Machine Learning Server](#)
- [Feature announcements from previous release of R Server](#)
- [Deprecated, discontinued or changed features](#)

Set up a data-science client for R development on SQL Server

4/12/2018 • 5 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

After you have configured an instance of SQL Server 2017 to support machine learning, you should set up a development environment that is capable of connecting to the server for remote execution and deployment.

This article describes some typical client scenarios, including configuration of the free Visual Studio Community edition to run R code in SQL Server.

Install R libraries on the client

Your client environment must include Microsoft R Open, as well as the additional RevoScaleR packages that support distributed execution of R on SQL Server. Standard distributions of R do not have the packages that support remote compute contexts or parallel execution of R tasks.

To get these libraries, install any of the following:

- [Microsoft R Client](#)
- Microsoft R Server (for SQL Server 2016)
 - To install from SQL Server setup, see [Install SQL Server 2016 R Server \(Standalone\)](#)
 - To use the separate Windows-based installer, see [Install Machine Learning Server for Windows](#)
- Machine Learning Server (for SQL Server 2017)
 - To install from SQL Server setup, see [Install SQL Server 2017 Machine Learning Server \(Standalone\)](#)
 - To use the separate Windows-based installer, see [Install R Server 9.1 for Windows](#)

R tools

When you install R with SQL Server, you get the same R tools that are installed with any **base** installation of R, such as RGui, Rterm, and so forth. Therefore technically, you have all the tools you need to develop and test R code.

The following standard R tools are included in a *base installation* of R, and therefore are installed by default.

- **RTerm:** A command-line terminal for running R scripts
- **RGui.exe:** A simple interactive editor for R. The command-line arguments are the same for RGui.exe and RTerm.
- **RScript:** A command-line tool for running R scripts in batch mode.

To locate these tools, determine the R library that was installed when you set up SQL Server or the standalone machine learning feature. For example, in a default installation, the R tools are located in these folders:

- SQL Server 2016 R Services: `~\Program Files\Microsoft SQL Server\MSSQL13.<instancename>\R_SERVICES\bin\x64`

- Microsoft R Server Standalone: `~\Program Files\Microsoft R\R_SERVER\bin\x64`
- SQL Server 2017 Machine Learning Services: `~\Program Files\Microsoft SQL Server\MSSQL14.<instancename>\R_SERVICES\bin\x64`
- Machine Learning Server (Standalone): `~\Program Files\Microsoft\ML Server\R_SERVER\bin\x64`

If you need help with the R tools, just open **RGui**, click **Help**, and select one of the options

Microsoft R Client

Microsoft R Client is a free download that gives you access to the RevoScaleR packages for development use. By installing R Client, you can create R solutions that can be run in all supported compute contexts, including SQL Server in-database analytics, and distributed R computing on Hadoop, Spark, or Linux using Machine Learning Server.

If you have already installed a different R development environment, such as RStudio, be sure to reconfigure the environment to use the libraries and executables provided by Microsoft R Client. By doing so you can use all the features of the RevoScaleR package, although performance will be limited.

For more information, see [What is Microsoft R Client?](#)

Install a development environment

If you don't already have a preferred R development environment, we recommend one of the following:

- R Tools for Visual Studio

Works with Visual Studio 2015.

For setup information, see [How to install R Tools for Visual Studio](#).

To configure RTVS to use your Microsoft R client libraries, see [About Microsoft R Client](#)

- Visual Studio 2017

Even the free Community Edition includes the data science workload, which installs project templates for R, Python, and F#.

Download Visual Studio from [this site](#).

- RStudio

If you prefer to use RStudio, some additional steps are required to use the RevoScaleR libraries:

- Install Microsoft R Client to get the required packages and libraries.
- Update your R path to use the Microsoft R runtime.

For more information, see [R Client - configure your IDE](#).

Configure your IDE

- R Tools for Visual Studio

See [this site](#) for some examples of how to build and debug R projects using R Tools for Visual Studio.

- Visual Studio 2017

If you install Microsoft R Client or R Server **before** you install Visual Studio, the R Server libraries are automatically detected and used for your library path. If you have not installed the RevoScaleR libraries, from the **R Tools** menu, select **Install R Client**.

Run R in SQL Server

This example uses Visual Studio 2017 Community Edition, with the data science workload installed.

1. From the **File** menu, select **New** and then select **Project**.
2. The -hand pane contains a list of preinstalled templates. Click **R**, and select **R Project**. In the **Name** box, type `dbtest` and click **OK**.
3. Visual Studio creates a new project folder and a default script file, `Script.R`.
4. Type `.libPaths()` on the first line of the script file, and then press CTRL + ENTER.
5. The current R library path should be displayed in the **R Interactive** window.
6. Click the **R Tools** menu and select **Windows** to see a list of other R-specific windows that you can display in your workspace.
 - View help on packages in the current library by pressing CTRL + 3.
 - See R variables in the **Variable Explorer**, by pressing CTRL + 8.
7. Create a connection string to a SQL Server instance, and use the connection string in the `RxInSqlServer` constructor to create a SQL Server data source object.

```
connStr <- "Driver=SQL Server;Server=MyServer;Database=MyTestDB;Uid=;Pwd="
sqlShareDir <- paste("C:\\AllShare\\", Sys.getenv("USERNAME"), sep = "")
sqlWait <- TRUE
sqlConsoleOutput <- FALSE
cc <- RxInSqlServer(connectionString = connStr, shareDir = sqlShareDir, wait = sqlWait, consoleOutput =
sqlConsoleOutput)
sampleDataQuery <- "SELECT TOP 100 * from [dbo].[MyTestTable]"
inDataSource <- RxSqlServerData(sqlQuery = sampleDataQuery, connectionString = connStr, rowsPerRead =
500)
```

TIP

To run a batch, select the lines you want to run and press CTRL + ENTER.

8. Set the compute context to the server, and then run some simple R code on the data.

```
rxSetComputeContext(cc)
rxGetVarInfo(data = inDataSource)
```

Results are returned in the **R Interactive** window.

If you want to assure yourself that the code is being executed on the SQL Server instance, you can use Profiler to create a trace.

Set up Python client tools for use with SQL Server Machine Learning

4/11/2018 • 8 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to configure a Python environment on a Windows computer to support running Python code in SQL Server. This includes the following scenarios:

- You test and develop solutions on a remote Python workstation, and send code to SQL Server for execution in a SQL Server compute context.

You generally want to install a full-featured Python development environment.

Your local Python environment should be compatible with the Python environment on the SQL Server instance, and you must install libraries that support remote compute contexts.

- You develop and test your code using dedicated Python tools, and then migrate the code to SQL Server to run in a stored procedure.

You use a full-featured Python IDE for development, off server. However, before you deploy your code, you test it in an environment that emulates the SQL Server environment.

- You primarily run Python code in a stored procedure in SQL Server and you do not develop Python code. You expect that the code has been tested and debugged before you wrap it in a stored procedure. However, occasionally you might need to run the code outside SQL Server, to determine the source of a problem.

You do not wish to install any Python tools on the server, and will use only the default tools installed with the distribution. You might decide to configure a Python environment that uses the instance library, to verify that code works outside of a stored procedure.

Requirements

Regardless of which tools you use to develop your Python code, the following requirements apply whenever you run Python code in SQL Server or use SQL Server data:

- To use Python, SQL Server 2017 or later is required. You must also install the feature that supports Machine Learning Services (In-Database), and explicitly enable the feature. For more information, see [Set up SQL Server 2017](#).

If you installed an early release of SQL Server 2017, you might get errors if you try to run Python commands from this command line utility. We recommend that you [upgrade to the latest version](#) if possible.

- You must ensure that the account you use to run the code has permission to connect to the database and to run Python code. The special permission `EXECUTE ANY EXTERNAL SCRIPT` is required for all accounts that use R or Python script.
- You must ensure that the account has any database permissions that might be required to read data or create new objects.
- If your code requires packages that are not installed by default with SQL Server, arrange with the database administrator to have the packages installed in the Python environment that is used by the instance. SQL Server is a secured environment and there are restrictions on where packages can be installed.

Ad hoc installation of packages as part of your code is not recommended, even if you have rights. Also, always carefully consider the security implications before installing new packages in the server library.

NOTE

If you intend to use Python with Machine Learning Server, which supports additional compute contexts, such as Linux or Spark clusters, see these articles:

- [How to install custom Python packages and interpreter locally on Windows](#)
- [Link Python tools and IDEs to the Python interpreter installed with Machine Learning Server](#)

Default tools included with standard install

A default installation of SQL Server 2017 with Machine Learning Services (In-database) and Python adds the following standard Python tools and resources:

- Python sample data
- Anaconda 4.2 distribution
- Python executables python.exe and pythonw.exe

By default, installation is to this folder: `~\Program Files\Microsoft SQL Server\<instance_name>\PYTHON_SERVICES`.

Open Python from the command line

To use the Python runtime that is installed with the instance, you can start the Python executable from the installation path. Basic instructions are available on the [Python for Windows FAQ](#).

IMPORTANT

Generally, to avoid resource contention, we recommend that you **do not** run Python from the instance library on the server, if you think it is possible the SQL Server instance is running Python code. However, using the tools in the instance library might be valuable if you are trying to debug an issue that occurs only when running in SQL Server and want to view more detailed error messages, or make sure that all required packages are installed.

1. Navigate to the folder where the instance library is installed. For example, in a default installation, the folder is `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES`.
2. Locate Python.exe.
3. Right-click and select **Run as administrator** to open an interactive command-line window.

Update Python components

You can update the Python components by downloading and installing a more recent version, using the script described here: [Install Python client on Windows](#)

The installer downloaded by the script is `SPO_9.3.0.0_1033.cab`. If you need a different version, see [Installing machine learning components without internet access](#)

Set up a Python development environment

If you are simply debugging scripts from the command line, you can get by with the standard Python tools installed with Machine Learning Services, or use a text editor. However, if you are developing new solutions, or working from a remote client, we recommend use of a full-featured Python IDE. Popular options are:

- [Visual Studio 2017 Community Edition](#) with Python
- [AI tools for Visual Studio](#)
- [Python in Visual Studio Code](#)
- Popular third party tools such as PyCharm, Spyder, and Eclipse

We recommend Visual Studio because it supports database projects as well as machine learning projects. For help configuring a Python environment, see [Managing Python environments in Visual Studio](#).

Install revoscalepy

Even if you have successfully installed Machine Learning Services, you might get an error when attempting to load **revoscalepy** functions from a Python command line. These steps describe how you can install an update to enable use of **revoscalepy**.

1. Download the installation shell script from <https://aka.ms/mls93-py> (or use <https://aka.ms/mls-py> for the 9.2. release). The script installs Anaconda 4.2.0, which includes Python 3.5.2, along with all packages listed previously.
2. Open a new PowerShell window with elevated permissions (as administrator).
3. Open the folder in which you downloaded the installer and run the script:

```
cd {{download-directory}}
.\Install-PyForMLS.ps1
```

You can also run the `-InstallFolder` command-line argument and specify the new path as part of the command. For example:

```
.\Install-PyForMLS.ps1 -InstallFolder "<installation_path>"
```

If you get an error, you might need to suspend execution policies for a specific file, the duration of the session, as follows:

```
powershell -ExecutionPolicy Bypass -File "C:\<installation_path>\Install-PyForMLS.ps1"
```

You can also suspend execution policies for the duration of the session. With this statement, the execution policy is set to `unrestricted` for the duration of the session, and does not change the configuration or write the change to disk.

```
Set-ExecutionPolicy Bypass -Scope Process
```

Verify that Python and revoscalepy are working

After installing all tools and libraries, you should connect to the server and verify that you can create a compute context, or that Python can communicate with SQL Server.

Verify that revoscalepy works from the Python command line

To verify that the **revoscalepy** module can be loaded, run the following sample code from a Python interactive command prompt. The code generates a data summary using the Python sample data and [rx_summary](#).

```

import os
from revoscalepy import rx_summary
from revoscalepy import RxXdfData
from revoscalepy import RxOptions
sample_data_path = RxOptions.get_option("sampleDataDir")
print(sample_data_path)
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)

```

The sample data path is printed so that you can determine which instance of Python is being called.

Verify that Python can be called from SQL Server

To verify that Python is communicating with SQL Server, open SQL Server Management Studio. (You can use another similar tool, such as [SQL Operations Studio](#).) Open a new **Query** window and run any simple Python command in the context of a stored procedure:

```

EXEC sp_execute_external_script @language = N'Python',
@script = N'print(3+4)'

```

It can take a while to launch the Python run-time for the first time, but if there are no errors, you know that the SQL Server Launchpad is working, and Python can be started from SQL Server.

To verify that **revoscalepy** is available in the SQL Server instance library, run the script based on [rx_summary](#), with some slight modifications to generate results compatible with SQL Server.

```

EXEC sp_execute_external_script @language = N'Python',
@script = N'
import os
from pandas import DataFrame
from revoscalepy import rx_summary
from revoscalepy import RxXdfData
from revoscalepy import RxOptions

sample_data_path = RxOptions.get_option("sampleDataDir")
print(sample_data_path)

ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay + DayOfWeek", ds)
print(summary)
dfsummary = summary.summary_data_frame
OutputDataSet = dfsummary
'

WITH RESULT SETS ((ColName nvarchar(25) , ColMean float, ColStdDev float, ColMin float, ColMax float,
Col_ValidObs float, Col_MissingObs int))

```

Because `rx_summary` returns an object of type `class revoscalepy.functions.RxSummary.RxSummaryResults`, which contains multiple elements, to handle the results in SQL Server, you can extract just the data frame in a tabular format.

Verify Python execution in SQL Server as remote compute context

If you have installed **revoscalepy** in your local Python development environment, you should be able to connect to an instance of SQL Server 2017 where Python has been enabled, and run a similar code sample using the server as the compute context.

```
import os
from revoscalepy import rx_summary, RxOptions, RxXdfData, RxSqlServerData, RxInSqlServer

# define connection string and compute context
sql_conn_string="Driver=SQL Server;Server=;Database=TestDB;Trusted_Connection=True"
sqlcc = RxInSqlServer(
    connection_string = sql_conn_string,
    num_tasks = 1,
    auto_cleanup = False,
    console_output = True,
    execution_timeout_seconds = 0,
    wait = True
)
rx_set_compute_context(sqlcc)

# get sample data and path
sample_data_path = RxOptions.get_option("sampleDataDir")
print(sample_data_path)

# generate summary
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)
```

In this sample, the summary object is returned to the console, rather than returning well-formed tabular data to SQL Server.

Also, because [rx_set_compute_context](#) has been invoked, the sample data is loaded from the samples folder on the SQL Server computer, and not from your local samples folder.

Quickstart: "Hello world" R script in SQL Server

7/16/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

SQL Server includes R language feature support for in-database analytics on resident SQL Server data. You can use open-source R functions, third-party packages, and built-in Microsoft R packages for predictive analytics at scale.

In this quickstart, you learn key concepts by running a "Hello World" R script in T-SQL, with an introduction to the **sp_execute_external_script** system stored procedure. R script execution is through stored procedures. You can either use the **sp_execute_external_script** stored procedure and pass R script in as an input parameter as demonstrated in this quickstart, or wrap R script in a [custom stored procedure](#).

Prerequisites

This exercise requires access to an instance of SQL Server with one of the following already installed:

- [SQL Server 2017 Machine Learning Services](#), with the R language installed
- [SQL Server 2016 R Services](#)

Your SQL Server instance can be in an Azure virtual machine or on-premises. Just be aware that the external scripting feature is disabled by default, so you might need to [enable external scripting](#) and verify that **SQL Server Launchpad service** is running before you start.

- A tool for running SQL queries. You can use any application that can connect to a SQL Server database and run T-SQL code. SQL professionals can use SQL Server Management Studio (SSMS) or Visual Studio.

For this tutorial, to show how easy it is to run R inside SQL Server, we've used the new **mssql extension for Visual Studio Code**. VS Code is a free development environment that can run on Linux, macOS, or Windows. The **mssql** extension is a lightweight extension for running T-SQL queries. To get Visual Studio Code, see [Download and install Visual Studio Code](#). To add the **mssql** extension, see this article: [Use the mssql extension for Visual Studio Code](#).

Connect to a database and run a Hello World test script

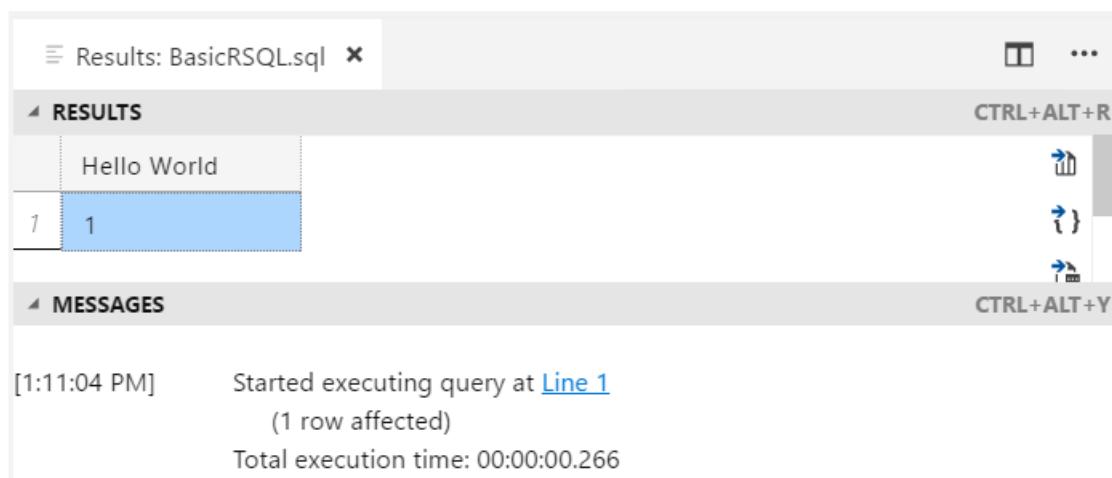
1. In Visual Studio Code, create a new text file and name it BasicRSQl.sql.
2. While this file is open, press CTRL+SHIFT+P (COMMAND + P on a macOS), type **sql** to list the SQL commands, and select **CONNECT**. Visual Studio Code prompts you to create a profile to use when connecting to a specific database. This is optional, but makes it easier to switch between databases and logins.
 - Choose a server or instance where R in SQL Server has been installed.
 - Use an account that has permissions to create a new database, run SELECT statements, and view table definitions.
3. If the connection is successful, you should be able to see the server and database name in the status bar, together with your current credentials. If the connection failed, check whether the computer name and server name are correct.
4. Paste in this statement and run it.

```
EXEC sp_execute_external_script
    @language =N'R',
    @script=N'OutputDataSet<-InputDataSet',
    @input_data_1 =N'SELECT 1 AS hello'
    WITH RESULT SETS (([Hello World] int));
GO
```

Inputs to this stored procedure include:

- *@language* parameter defines the language extension to call, in this case, R.
- *@script* parameter defines the commands passed to the R runtime. Your entire R script must be enclosed in this argument, as Unicode text. You could also add the text to a variable of type **nvarchar** and then call the variable.
- *@input_data_1* is data returned by the query, passed to the R runtime, which returns the data to SQL Server as a data frame.
- **WITH RESULT SETS** clause defines the schema of the returned data table for SQL Server, adding "Hello World" as the column name, **int** for the data type.

Results



The screenshot shows the SSMS Results pane for a query named 'BasicRSQl.sql'. The results section displays a single row with the value 'Hello World' in the first column and the number '1' in the second column. The messages section shows the following log:

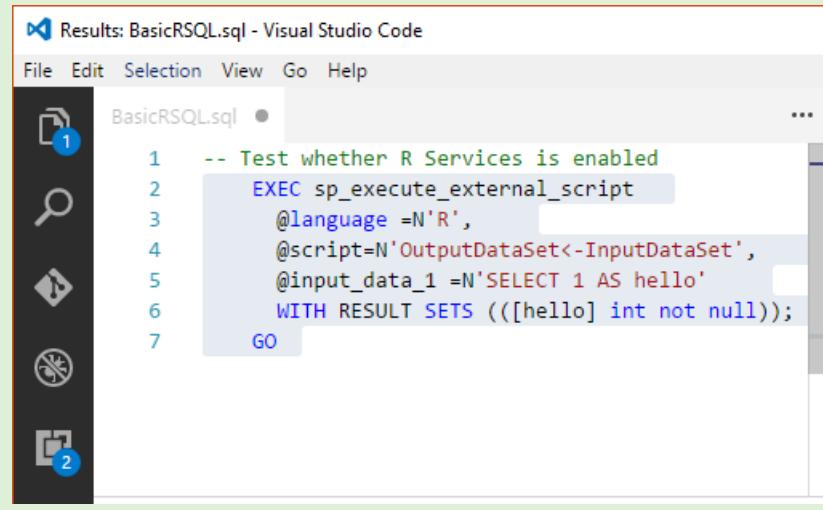
```
[1:11:04 PM]      Started executing query at Line 1
                  (1 row affected)
Total execution time: 00:00:00.266
```

If you get any errors from this query, rule out any installation issues. Post-install configuration is required to enable use of external code libraries. See [Install SQL Server 2017 Machine Learning Services](#) or [Install SQL Server 2016 R Services](#). Likewise, make sure that the Launchpad service is running.

Depending on your environment, you might need to enable the R worker accounts to connect to SQL Server, install additional network libraries, enable remote code execution, or restart the instance after everything is configured. For more information, see [R Services Installation and Upgrade FAQ](#)

TIP

In Visual Studio Code, you can highlight the code you want to run and press CTRL+SHIFT+E. If this is too hard to remember, you can change it! See [Customize the shortcut key bindings](#).



```
1 -- Test whether R Services is enabled
2 EXEC sp_execute_external_script
3   @language =N'R',
4   @script=N'OutputDataSet<-InputDataSet',
5   @input_data_1 =N'SELECT 1 AS hello'
6   WITH RESULT SETS (([hello] int not null));
7 GO
```

Next steps

Now that you have confirmed your instance is ready to work with R, take a closer look at structuring inputs and outputs.

[Quickstart: Handle inputs and outputs](#)

Quickstart: Handle inputs and outputs using R in SQL Server

7/16/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

When you want to run R code in SQL Server, you must wrap R script in a stored procedure. You can write one, or pass R script to [sp_execute_external_script](#). This system stored procedure is used to start the R runtime in the context of SQL Server, which passes data to R, manages R user sessions securely, and returns any results to the client.

Prerequisites

A previous quickstart, [Hello World in R and SQL](#), provides information and links for setting up the R environment required for this quickstart.

Create some simple test data

Create a small table of test data by running the following T-SQL statement:

```
CREATE TABLE RTestData ([col1] int not null) ON [PRIMARY]
INSERT INTO RTestData    VALUES (1);
INSERT INTO RTestData    VALUES (10);
INSERT INTO RTestData    VALUES (100) ;
GO
```

When the table has been created, use the following statement to query the table:

```
SELECT * FROM RTestData
```

Results

RESULTS	
	col1
1	1
2	10
3	100

Get the same data using R script

After the table has been created, run the following statement:

```

EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' OutputDataSet <- InputDataSet;'
    , @input_data_1 = N' SELECT * FROM RTestData;'
    WITH RESULT SETS (([NewColName] int NOT NULL));

```

It gets the data from the table, makes a round trip through the R runtime, and returns the values with the column name, *NewColName*.

Results

	NewColName
1	1
2	10
3	100

[11:07:06 AM] Started executing query at [Line 18](#)
(3 rows affected)
Total execution time: 00:00:03.093

Comments

- The `@language` parameter defines the language extension to call, in this case, R.
- In the `@script` parameter, you define the commands to pass to the R runtime. Your entire R script must be enclosed in this argument, as Unicode text. You could also add the text to a variable of type **nvarchar** and then call the variable.
- The data returned by the query is passed to the R runtime, which returns the data to SQL Server as a data frame.
- The `WITH RESULT SETS` clause defines the schema of the returned data table for SQL Server.

Change input or output variables

The preceding example used the default input and output variable names, *InputDataSet* and *OutputDataSet*. To define the input data associated with *InputDataSet*, you use the `@input_data_1` variable.

In this example, the names of the output and input variables for the stored procedure have been changed to *SQL_Out* and *SQL_In*:

```

EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' SQL_out <- SQL_in;'
    , @input_data_1 = N' SELECT 12 as Col;'
    , @input_data_1_name = N'SQL_In'
    , @output_data_1_name = N'SQL_Out'
    WITH RESULT SETS (([NewColName] int NOT NULL));

```

Did you get the error, "object SQL_in not found"? That's because R is case-sensitive! In the example, the R script uses the variables *SQL_in* and *SQL_out*, but the parameters to the stored procedure use the variables *SQL_In* and *SQL_Out*.

Try correcting **only** the *SQL_In* variable to `@script` and re-run the stored procedure.

Now you get a **different** error:

```
EXECUTE statement failed because its WITH RESULT SETS clause specified 1 result set(s), but the statement only sent 0 result set(s) at run time.
```

We're showing you this error because you can expect to see it often when testing new R code. It means that the R script ran successfully, but SQL Server received no data, or received wrong or unexpected data.

In this case, the output schema (the line beginning with **WITH**) specifies that one column of integer data should be returned, but since R put the data in a different variable, nothing came back to SQL Server; hence, the error. To fix the error, correct the second variable name.

Remember these requirements!

- Variable names must follow the rules for valid SQL identifiers.
- The order of the parameters is important. You must specify the required parameters `@input_data_1` and `@output_data_1` first, in order to use the optional parameters `@input_data_1_name` and `@output_data_1_name`.
- Only one input dataset can be passed as a parameter, and you can return only one dataset. However, you can call other datasets from inside your R code and you can return outputs of other types in addition to the dataset. You can also add the `OUTPUT` keyword to any parameter to have it returned with the results. There is a simple example later in this tutorial.
- The `WITH RESULT SETS` statement defines the schema for the data, for the benefit of SQL Server. You need to provide SQL compatible data types for each column you return from R. You can use the schema definition to provide new column names too; you need not use the column names from the R `data.frame`. In some cases, this clause is optional; try omitting it and see what happens.

Generate results using R

You can also generate values using just the R script and leave the input query string in `@input_data_1` blank. Or, use a valid SQL `SELECT` statement as a placeholder, and not use the SQL results in the R script.

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' mytextvariable <- c("hello", " ", "world");
        OutputDataSet <- as.data.frame(mytextvariable);'
    , @input_data_1 = N' SELECT 1 as Temp1'
    WITH RESULT SETS (([Col1] char(20) NOT NULL));
```

Results

RESULTS	
	Col1
1	hello
2	
3	world

Next steps

Examine some of the problems that you might encounter when passing data between R and SQL Server, such as implicit conversions and differences in tabular data between R and SQL.

[Quickstart: Handle data types and objects](#)

Quickstart: Handle data types and objects using R in SQL Server

7/16/2018 • 9 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this quickstart, get a hands-on introduction to common issues that occur when moving data between R and SQL Server. The experience you gain through this exercise provides essential background when working with data in your own script.

Common issues to know up front include:

- Data types sometimes do not match
- Implicit conversions might take place
- Cast and convert operations are sometimes required
- R and SQL use different data objects

Prerequisites

A previous quickstart, [Hello World in R and SQL](#), provides information and links for setting up the R environment required for this quickstart.

Always return a data frame

When your script returns results from R to SQL Server, it must return the data as a **data.frame**. Any other type of object that you generate in your script — whether that be a list, factor, vector, or binary data — must be converted to a data frame if you want to output it as part of the stored procedure results. Fortunately, there are multiple R functions to support changing other objects to a data frame. You can even serialize a binary model and return it in a data frame, which you'll do later in this tutorial.

First, let's experiment with some R basic R objects — vectors, matrices, and lists — and see how conversion to a data frame changes the output passed to SQL Server.

Compare these two "Hello World" scripts in R. The scripts look almost identical, but the first returns a single column of three values, whereas the second returns three columns with a single value each.

Example 1

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' mytextvariable <- c("hello", " ", "world");
        OutputDataSet <- as.data.frame(mytextvariable);'
    , @input_data_1 = N' ';
```

Example 2

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' OutputDataSet<- data.frame(c("hello"), " ", c("world"));'
    , @input_data_1 = N' ';
```

Identify schema and data types

Why are the results so different?

The answer can usually be found by using the R `str()` command. Add the function `str(object_name)` anywhere in your R script to have the data schema of the specified R object returned as an informational message. To view messages, see in the **Messages** pane of Visual Studio Code, or the **Messages** tab in SSMS.

To figure out why Example 1 and Example 2 have such different results, insert the line `str(OutputDataSet)` at the end of the `@script` variable definition in each statement, like this:

Example 1 with str function added

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N' mytextvariable <- c("hello", " ", "world");
        OutputDataSet <- as.data.frame(mytextvariable);
        str(OutputDataSet);'
    , @input_data_1 = N'  '
;
```

Example 2 with str function added

```
EXECUTE sp_execute_external_script
    @language = N'R',
    @script = N' OutputDataSet <- data.frame(c("hello"), " ", c("world"));
        str(OutputDataSet);',
    @input_data_1 = N'  ';
```

Now, review the text in **Messages** to see why the output is different.

Results - Example 1

```
STDOUT message(s) from external script:
'data.frame': 3 obs. of 1 variable:
$ mytextvariable: Factor w/ 3 levels "","hello","world": 2 1 3
```

Results - Example 2

```
STDOUT message(s) from external script:
'data.frame': 1 obs. of 3 variables:
$ c..hello...: Factor w/ 1 level "hello": 1
$ X...       : Factor w/ 1 level " ": 1
$ c..world...: Factor w/ 1 level "world": 1
```

As you can see, a slight change in R syntax had a big effect on the schema of the results. We won't go into why, because the differences in R data types are explained more thoroughly in this article by Hadley Wickham: [R Data Structures](#).

For now, just be aware that you need to check the expected results when coercing R objects into data frames.

TIP

You can also use R identity functions, such as `is.matrix`, `is.vector`, to return information about the internal data structure.

Implicit conversion of data objects

Each R data object has its own rules for how values are handled when combined with other data objects if the two data objects have the same number of dimensions, or if any data object contains heterogeneous data types.

For example, assume you run the following statement to perform matrix multiplication using R. You multiply a single-column matrix with three values by an array with four values, and expect a 4x3 matrix as a result.

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
        x <- as.matrix(ResultSet);
        y <- array(12:15);
        OutputDataSet <- as.data.frame(x %*% y);
        , @input_data_1 = N' SELECT [Col1] from RTestData;
        WITH RESULT SETS (([Col1] int, [Col2] int, [Col3] int, Col4 int));
```

Under the covers, the column of three values is converted to a single-column matrix. Because a matrix is just a special case of an array in R, the array `y` is implicitly coerced to a single-column matrix to make the two arguments conform.

Results

COL1	COL2	COL3	COL4
12	13	14	15
120	130	140	150
1200	1300	1400	1500

However, note what happens when you change the size of the array `y`.

```
execute sp_execute_external_script
    @language = N'R'
    , @script = N'
        x <- as.matrix(ResultSet);
        y <- array(12:14);
        OutputDataSet <- as.data.frame(y %*% x);
        , @input_data_1 = N' SELECT [Col1] from RTestData;
        WITH RESULT SETS (([Col1] int ));
```

Now R returns a single value as the result.

Results

COL1
1542

Why? In this case, because the two arguments can be handled as vectors of the same length, R returns the inner product as a matrix. This is the expected behavior according to the rules of linear algebra; however, it could cause problems if your downstream application expects the output schema to never change!

TIP

Getting errors? These examples require the table **RTestData**. If you haven't created the test data table, go back to this topic to create the table: [Handle inputs and outputs](#).

If you have created the table but still get an error, make sure that you are running the stored procedure in the context of the database that contains the table, and not in **master** or another database.

Also, we suggest that you avoid using temporary tables for these examples. Some R clients will terminate a connection between batches, deleting temporary tables.

Merge or multiply columns of different length

R provides great flexibility for working with vectors of different sizes, and for combining these column-like structures into data frames. Lists of vectors can look like a table, but they don't follow all the rules that govern database tables.

For example, the following script defines a numeric array of length 6 and stores it in the R variable `df1`. The numeric array is then combined with the integers of the **RTestData** table, which contains three (3) values, to make a new data frame, `df2`.

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
        df1 <- as.data.frame( array(1:6) );
        df2 <- as.data.frame( c( InputDataSet , df1 ) );
        OutputDataSet <- df2'
    , @input_data_1 = N' SELECT [Col1]  from RTestData;'
    WITH RESULT SETS (( [Col2] int not null, [Col3] int not null ));
```

To fill out the data frame, R repeats the elements retrieved from **RTestData** as many times as needed to match the number of elements in the array `df1`.

Results

COL2	COL3
1	1
10	2
100	3
1	4
10	5
100	6

Remember that a data frame only looks like a table, and is actually a list of vectors.

Cast or convert SQL Server data

R and SQL Server don't use the same data types, so when you run a query in SQL Server to get data and then pass that to the R runtime, some type of implicit conversion usually takes place. Another set of conversions takes

place when you return data from R to SQL Server.

- SQL Server pushes the data from the query to the R process managed by the Launchpad service and converts it to an internal representation for greater efficiency.
- The R runtime loads the data into a data.frame variable and performs its own operations on the data.
- The database engine returns the data to SQL Server using a secured internal connection and presents the data in terms of SQL Server data types.
- You get the data by connecting to SQL Server using a client or network library that can issue SQL queries and handle tabular data sets. This client application can potentially affect the data in other ways.

To see how this works, run a query such as this one on the [AdventureWorksDW](#) data warehouse. This view returns sales data used in creating forecasts.

```
USE AdventureWorksDW
GO

SELECT ReportingDate
      , CAST(ModelRegion as varchar(50)) as ProductSeries
      , Amount
    FROM [AdventureWorksDW].[dbo].[vTimeSeries]
   WHERE [ModelRegion] = 'M200 Europe'
ORDER BY ReportingDate ASC
```

NOTE

You can use any version of AdventureWorks, or create a different query using a database of your own. The point is to try to handle some data that contains text, datetime and numeric values.

Now, try pasting this query as the input to the stored procedure.

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N' str(InputDataSet);
OutputDataSet <- InputDataSet;'
  , @input_data_1 = N'
    SELECT ReportingDate
      , CAST(ModelRegion as varchar(50)) as ProductSeries
      , Amount
    FROM [AdventureWorksDW].[dbo].[vTimeSeries]
   WHERE [ModelRegion] = ''M200 Europe''
ORDER BY ReportingDate ASC ;'
WITH RESULT SETS undefined;
```

If you get an error, you'll probably need to make some edits to the query text. For example, the string predicate in the WHERE clause must be enclosed by two sets of single quotation marks.

After you get the query working, review the results of the `str` function to see how R treats the input data.

Results

```
STDOUT message(s) from external script: 'data.frame': 37 obs. of 3 variables:
STDOUT message(s) from external script: $ ReportingDate: POSIXct, format: "2010-12-24 23:00:00" "2010-12-24
23:00:00"
STDOUT message(s) from external script: $ ProductSeries: Factor w/ 1 levels "M200 Europe",...: 1 1 1 1 1 1 1 1
1 1
STDOUT message(s) from external script: $ Amount : num 3400 16925 20350 16950 16950
```

- The datetime column has been processed using the R data type, **POSIXct**.
- The text column "ProductSeries" has been identified as a **factor**, meaning a categorical variable. String values are handled as factors by default. If you pass a string to R, it is converted to an integer for internal use, and then mapped back to the string on output.

Summary

From even these short examples, you can see the need to check the effects of data conversion when passing SQL queries as input. Because some SQL Server data types are not supported by R, consider these ways to avoid errors:

- Test your data in advance and verify columns or values in your schema that could be a problem when passed to R code.
- Specify columns in your input data source individually, rather than using `SELECT *`, and know how each column will be handled.
- Perform explicit casts as necessary when preparing your input data, to avoid surprises.
- Avoid passing columns of data (such as GUIDS or rowguids) that cause errors and aren't useful for modeling.

For more information on supported and unsupported data types, see [R libraries and data types](#).

For information about the performance impact of run-time conversion of strings to numerical factors, see [SQL Server R Services performance tuning](#).

Next step

In the next quickstart, you'll learn how to apply R functions to SQL Server data.

[Quickstart: Use R functions with SQL Server data](#)

Quickstart: Create a predictive model using R in SQL Server

7/16/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this quickstart, you'll learn how to train a model using R, and then save the model to a table in SQL Server. The model is a simple regression model that predicts the stopping distance of a car based on speed. You'll use the `cars` dataset included with R, because it is small and easy to understand.

Prerequisites

A previous quickstart, [Hello World in R and SQL](#), provides information and links for setting up the R environment required for this quickstart.

Create the source data

First, create a table to save the training data.

```
CREATE TABLE CarSpeed ([speed] int not null, [distance] int not null)
INSERT INTO CarSpeed
EXEC sp_execute_external_script
    @language = N'R'
    , @script = N'car_speed <- cars;'
    , @input_data_1 = N''
    , @output_data_1_name = N'car_speed'
```

- Some people like to use temporary tables, but be aware that some R clients disconnect sessions between batches.
- Many datasets, small and large, are included with the R runtime. To get a list of datasets installed with R, type `library(help="datasets")` from an R command prompt.

Create a regression model

The car speed data contains two columns, both numeric, `dist` and `speed`. There are multiple observations of some speeds. From this data, you will create a linear regression model that describes some relationship between car speed and the distance required to stop a car.

The requirements of a linear model are simple:

- Define a formula that describes the relationship between the dependent variable `speed` and the independent variable `distance`
- Provide input data to use in training the model

TIP

If you need a refresher on linear models, we recommend this tutorial, which describes the process of fitting a model using rxLinMod: [Fitting Linear Models](#)

To actually build the model, you define the formula inside your R code, and pass the data as an input parameter.

```
DROP PROCEDURE IF EXISTS generate_linear_model;
GO
CREATE PROCEDURE generate_linear_model
AS
BEGIN
    EXEC sp_execute_external_script
        @language = N'R'
        , @script = N'lrm <- rxLinMod(formula = distance ~ speed, data = CarsData);
        trained_model <- data.frame(payload = as.raw(serializerserialize(lrm, connection=NULL)));
        , @input_data_1 = N'SELECT [speed], [distance] FROM CarSpeed'
        , @input_data_1_name = N'CarsData'
        , @output_data_1_name = N'trained_model'
        WITH RESULT SETS ((model varbinary(max)));
END;
GO
```

- The first argument to rxLinMod is the *formula* parameter, which defines distance as dependent on speed.
- The input data is stored in the variable `CarsData`, which is populated by the SQL query. If you don't assign a specific name to your input data, the default variable name would be *InputDataSet*.

Create a table for storing the model

Next, store the model so you can retrain or use it for prediction. The output of an R package that creates a model is usually a **binary object**. Therefore, the table where you store the model must provide a column of **varbinary** type.

```
CREATE TABLE stopping_distance_models (
    model_name varchar(30) not null default('default model') primary key,
    model varbinary(max) not null;
```

Save the model

To save the model, run the following Transact-SQL statement to call the stored procedure, generate the model, and save it to a table.

```
INSERT INTO stopping_distance_models (model)
EXEC generate_linear_model;
```

Note that if you run this code a second time, you get this error:

```
Violation of PRIMARY KEY constraint...Cannot insert duplicate key in object dbo.stopping_distance_models
```

One option for avoiding this error is to update the name for each new model. For example, you could change the name to something more descriptive, and include the model type, the day you created it, and so forth.

```
UPDATE stopping_distance_models
SET model_name = 'rxLinMod ' + format(getdate(), 'yyyy.MM.HH.mm', 'en-gb')
WHERE model_name = 'default model'
```

Output additional variables

Generally, the output of R from the stored procedure `sp_execute_external_script` is limited to a single data frame.

(This limitation might be removed in future.)

However, you can return outputs of other types, such as scalars, in addition to the data frame.

For example, suppose you want to train a model but immediately view a table of coefficients from the model. You could create the table of coefficients as the main result set, and output the trained model in a SQL variable. You could immediately re-use the model by calling the variable, or you could save the model to a table as shown here.

```
DECLARE @model varbinary(max), @modelname varchar(30)
EXEC sp_execute_external_script
    @language = N'R'
    , @script = N'
        speedmodel <- rxLinMod(distance ~ speed, CarsData)
        modelbin <- serialize(speedmodel, NULL)
        OutputDataSet <- data.frame(coefficients(speedmodel));
    , @input_data_1 = N'SELECT [speed], [distance] FROM CarSpeed'
    , @input_data_1_name = N'CarsData'
    , @params = N'@modelbin varbinary(max) OUTPUT'
    , @modelbin = @model OUTPUT
    WITH RESULT SETS (([Coefficient] float not null));

-- Save the generated model
INSERT INTO [dbo].[stopping_distance_models] (model_name, model)
VALUES ('latest model', @model)
```

Results

RESULTS	
	Coefficient
1	-17.579094890...
2	3.93240875912...

Summary

Remember these rules for working with SQL parameters and R variables in `sp_execute_external_script`:

- All SQL parameters mapped to R script must be listed by name in the `@params` argument.
- To output one of these parameters, add the `OUTPUT` keyword in the `@params` list.
- After listing the mapped parameters, provide the mapping, line by line, of SQL parameters to R variables, immediately after the `@params` list.

Next steps

Now that you have a model, in the final quickstart, you'll learn how to generate predictions from it and plot the results.

[Quickstart: Predict and plot from model](#)

Quickstart: Predict and plot from model using R in SQL Server

7/16/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

In this quickstart, use the model you created in the previous quickstart to score predictions against fresh data. To perform *scoring* using new data, get one of the trained models from the table, and then call a new set of data on which to base predictions. Scoring is a term sometimes used in data science to mean generating predictions, probabilities, or other values based on new data fed into a trained model.

Prerequisites

This quickstart is an extension of [Create a predictive model](#).

Create the table of new speeds

Did you notice that the original training data stops at a speed of 25 miles per hour? That's because the original data was based on an experiment from 1920!

You might wonder, how long would it take an automobile from the 1920s to stop, assuming it could get going as fast as 60 mph or even 100 mph? To answer this question, you must provide some new speed values.

```
CREATE TABLE [dbo].[NewCarSpeed]([speed] [int] NOT NULL,
    [distance] [int] NULL) ON [PRIMARY]
GO
INSERT [dbo].[NewCarSpeed] (speed)
VALUES (40), (50), (60), (70), (80), (90), (100)
```

Predict stopping distance

By now, your table might contain multiple R models, all built using different parameters or algorithms, or trained on different subsets of data.

RESULTS	
	model_name
1	default model
2	rxLinMod 2017.02.12.48
3	rxLinMod 2017.02.12.50
4	rxLinMod 2017.02.12.51
5	rxLinMod 2017.02.14.57

MESSAGES	
[3:07:47 PM]	Started executing query at Line 67 (5 rows affected) Total execution time: 00:00:00.016

To get predictions based on one specific model, you must write a SQL script that does the following:

1. Gets the model you want
2. Gets the new input data
3. Calls an R prediction function that is compatible with that model

In this example, because your model is based on the **rxLinMod** algorithm provided as part of the **RevoScaleR** package, you call the **rxPredict** function, rather than the generic R **predict** function.

```
DECLARE @speedmodel varbinary(max) = (SELECT model FROM [dbo].[stopping_distance_models] WHERE model_name = 'latest model');
EXEC sp_execute_external_script
    @language = N'R'
    , @script = N'
        current_model <- unserialize(as.raw(speedmodel));
        new <- data.frame(NewCarData);
        predicted.distance <- rxPredict(current_model, new);
        str(predicted.distance);
        OutputDataSet <- cbind(new, ceiling(predicted.distance));

        ,
        , @input_data_1 = N'SELECT speed FROM [dbo].[NewCarSpeed]'
        , @input_data_1_name = N'NewCarData'
        , @params = N'@speedmodel varbinary(max)'
        , @speedmodel = @speedmodel
WITH RESULT SETS (([new_speed] INT, [predicted_distance] INT))
```

- Use a SELECT statement to get a single model from the table, and pass it as an input parameter.
- After retrieving the model from the table, call the **unserialize** function on the model.

TIP

Also check out the new [serialization functions](#) provided by RevoScaleR, which support [realtime scoring](#).

- Apply the **rxPredict** function with appropriate arguments to the model, and provide the new input data.
- In the example, the **str** function is added during the testing phase, to check the schema of data being returned from R. You can remove the statement later.
- The column names used in the R script are not necessarily passed to the stored procedure output. Here we've used the WITH RESULTS clause to define some new column names.

Results

RESULTS

	new_speed	predicted_dista...
1	40	140
2	50	180
3	60	219
4	70	258
5	80	298
6	90	337
7	100	376

MESSAGES

```
[3:21:38 PM] Started executing query at Line 71
(7 rows affected)
STDOUT message(s) from external script:
Rows Read: 7, Total Rows Processed: 7, Total Chunk Time: 0.001 seconds
'data.frame': 7 obs. of 1 variable:
 $ distance_Pred: num 140 179 218 258 297 ...
Total execution time: 00:00:00.475
```

Perform scoring in parallel

The predictions came back fairly fast on this tiny data set. But suppose you needed to make lots of predictions very fast? There are many ways to speed up operations in SQL Server, more so if the operations can be processed in parallel. For scoring in particular, one easy way is to add the `@parallel` parameter to `sp_execute_external_script` and set the value to **1**.

Let's assume that you have obtained a much bigger table of possible car speeds, including hundreds of thousands of values. There are many sample T-SQL scripts from the community to help you generate number tables, so we won't reproduce those here. Let's just say that you have a column containing many integers, and want to use that as input for `speed` in the model.

To do this, just run the same prediction query, but substitute the larger dataset, and add the `@parallel = 1` argument.

```
DECLARE @speedmodel varbinary(max) = (select model from [dbo].[stopping_distance_models] where model_name =
'default_model');
EXEC sp_execute_external_script
    @language = N'R'
    , @script = N'
        current_model <- unserialize(as.raw(speedmodel));
        new <- data.frame(NewCarData);
        predicted.distance <- rxPredict(current_model, new);
        OutputDataSet <- cbind(new, ceiling(predicted.distance));

        ,
        , @input_data_1 = N' SELECT [speed] FROM [dbo].[HugeTableofCarSpeeds] '
        , @input_data_1_name = N'NewCarData'
        , @parallel = 1
        , @params = N'@speedmodel varbinary(max)'
        , @speedmodel = @speedmodel
WITH RESULT SETS (([new_speed] INT, [predicted_distance] INT))
```

- Parallel execution generally provides benefits only when working with very large data. The SQL database engine might decide that parallel execution is not needed. Moreover, the SQL query that gets your data must be capable of generating a parallel query plan.
- When using the option for parallel execution, you **must** specify the output results schema in advance, by using the `WITH RESULT SETS` clause. Specifying the output schema in advance allows SQL Server to

aggregate the results of multiple parallel datasets, which otherwise might have unknown schemas.

- If you are *training* a model instead of *scoring*, this parameter often won't have an effect. Depending on the model type, model creation might require that all the rows be read before summaries can be created.
- To get the benefits of parallel processing when you train your model, we recommend that you use one of the **RevoScaleR** algorithms. These algorithms are designed to distribute processing automatically, even if you don't specify `@parallel =1` in the call to `sp_execute_external_script`. For guidance on how to get the best performance with RevoScaleR algorithms, see [Distributed and parallel computing with ScaleR in Microsoft R](#).

Create an R plot of the model

Many clients, including SQL Server Management Studio, cannot directly display plots created using `sp_execute_external_script`. Instead, the general process for generating R plots is to create the plot as part of your R code and then write the image to a file.

Alternatively, you can return the serialized binary plot object to any application that can display images.

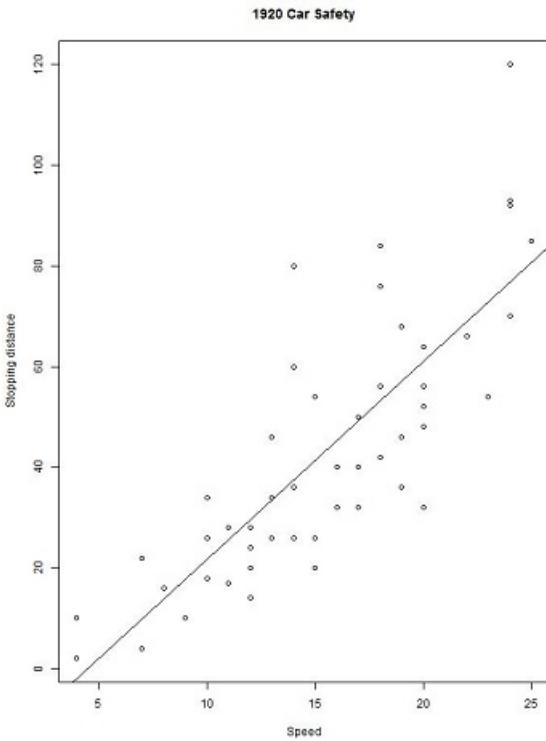
The following example demonstrates how to create a simple graphic using a plotting function included by default with R. The image is output to the specified file, and is also output into a SQL variable by the stored procedure.

```
EXECUTE sp_execute_external_script
@language = N'R'
, @script = N'
imageDir <- ''C:\\temp\\plots'';
image_filename = tempfile(pattern = "plot_", tmpdir = imageDir, fileext = ".jpg")
print(image_filename);
jpeg(filename=image_filename, width=600, height = 800);
print(plot(distance~speed, data=InputDataSet, xlab="Speed", ylab="Stopping distance", main = "1920 Car
Safety"));
abline(lm(distance~speed, data = InputDataSet));
dev.off();
OutputDataSet <- data.frame(data=readBin(file(image_filename, "rb"), what=raw(), n=1e6));
'

, @input_data_1 = N'SELECT speed, distance from [dbo].[CarSpeed]'
WITH RESULT SETS ((plot varbinary(max)));
```

- The `tempfile` function returns a string that can be used as a file name, but the file has not been generated yet.
- For arguments to `tempfile`, you can specify a prefix and file extension, as well as the directory. To verify the complete file name and path, print a message using `str()`.
- The `jpeg` function creates an R device with the specified parameters.
- After you create the plot, you can add more visual features to it. In this case, a regression line is added using `abline`.
- When you are done adding plot features, you must close the graphics device using the `dev.off()` function.
- The `readBin` function takes a file to read, a format specification, and the number of records. The `rb**`` keyword indicates that the file is binary rather than text.

Results



If you want to do some more elaborate plots, using some of the great graphics packages for R, we recommend these articles. Both require the popular **ggplot2** package.

- [Loan Classification using SQL Server 2016 R Services](#): End-to-end scenario based on insurance data. Requires the **reshape** package.
- [Create Graphs and Plots Using R](#)

Next steps

Integration of R with SQL Server makes it easier to deploy R solutions at scale, leveraging the best features of R and relational databases, for high-performance data handling and rapid R analytics.

Continue learning about solutions using R with SQL Server through end-to-end scenarios created by the Microsoft Data Science and R Services development teams.

[SQL Server R tutorials](#)

Tutorials for SQL Server Machine Learning Services

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides a comprehensive list of the tutorials, demos, and sample applications that use machine learning features in SQL Server 2016 or SQL Server 2017. Start here to learn how to run R or Python from T-SQL, how to use remote and local compute contexts, and how to optimize your R and Python code for a SQL production environment.

Start here

- [Python tutorials](#)
- [R tutorials](#)

For more information about requirements and how to get set up, see [Prerequisites](#).

Samples and solutions

- [Samples](#)

These real-world scenarios from the SQL Server development team demonstrate how to embed machine learning in applications. All samples include code that you can download, modify, and use in production.

- [Solutions](#)

Templates from the Microsoft Data Science team are customizable, to get you started fast with machine learning. Each solution is tailored to a specific task or industry problem. Most of the solutions are designed to run either in SQL Server, or in a cloud environment such as Azure Machine Learning. Other solutions can run on Linux or in Spark or Hadoop clusters, by using Microsoft R Server or Machine Learning Server.

SQL Server product samples

These samples and demos provided by the SQL Server and R Server development team highlight ways that you can use embedded analytics in real-world applications.

- [Perform customer clustering using R and SQL Server](#)

Use unsupervised learning to segment customers based on sales data. This example uses the scalable rxKmeans algorithm from Microsoft R to build the clustering model.

Applies to: SQL Server 2016 or SQL Server 2017

- [NEW! Perform customer clustering using Python and SQL Server](#)

Learn how to use the Kmeans algorithm to perform unsupervised clustering of customers. This example uses the Python language in-database.

Applies to: SQL Server 2017

- [Build a predictive model using R and SQL Server](#)

Learn how a ski rental business might use machine learning to predict future rentals, which helps the business plan and staff to meet future demand. This example uses the Microsoft algorithms to build

logistic regression and decision trees models.

Applies to: SQL Server 2016 or SQL Server 2017

- [Build a predictive model using Python and SQL Server](#)

Build the ski rental analysis application using Python, to help plan for future demand. This example uses the new Python library, **revoscalepy**, to create a linear regression model.

Applies to: SQL Server 2017

- [How to use Tableau with SQL Server Machine Learning Services](#)

Analyze social media and create Tableau graphs, using SQL Server and R.

Solution templates

The Microsoft Data Science Team has provided solution templates that can be used to jump-start solutions for common scenarios. All code is provided, along with instructions on how to train and deploy a model for scoring using SQL Server stored procedures.

- [Fraud detection](#)
- [Custom churn prediction](#)
- [Predictive maintenance](#)
- [Predict hospital length of stay](#)

For more information, see [Machine Learning Templates with SQL Server 2016 R Services](#).

More resources and reading

- [Why did we build it?](#)

Want to know the real story behind R Services? Read this article from the development and PM team that explains the origin and goals of SQL Server R Services.

- [Tutorials and sample data for Microsoft R](#)

Learn about Microsoft R, and what the RevoScaleR package offers in this collection of quick tutorials. Learn how to write R code once and deploy anywhere, using RevoScaleR data sources and remote compute contexts.

- [Get started with MicrosoftML](#)

Learn how to use the new algorithms in the MicrosoftML package for advanced modeling and scalable data transformations, optimized for multiple compute contexts.

Prerequisites

To run these tutorials, you must download and install the SQL Server machine learning components, as described here:

- [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#)
- [Install SQL Server 2016 R Services \(In-Database\)](#)

With SQL Server 2017, you can install either R or Python, or both. Otherwise the overall setup process, architecture, and requirements are the same.

After running SQL Server setup, don't forget these important steps:

1. Enable the external script execution feature by running `sp_configure 'external scripts enabled', 1`. Follow

the instructions to reconfigure and restart SQL Server.

2. Ensure that the Launchpad service is running, and that the Launchpad worker accounts can connect to the SQL Server instance.
3. Review the permissions associated with the users who must run R or Python scripts. Regardless of whether you use SQL logins or Windows user accounts, the user must have permission to run R or Python scripts, and must be able to connect to the instance. Depending on the tutorial, the user might also require permission to write data, create database objects, or do a bulk import of data.

For details, see this article for some common setup and configuration issues: [Troubleshooting Machine Learning Services](#)

NOTE

You cannot run these tutorials using another open source R or Python tool. Both your development environment and the SQL Server computer with machine learning must have the R or Python libraries provided by Microsoft, which support integration with SQL Server and use of remote compute contexts.

SQL Server R tutorials

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides a list of tutorials and samples that demonstrate the use of R with SQL Server 2016 or SQL Server 2017. Through these samples and demos, you will learn:

- How to run R from T-SQL
- What are remote and local compute contexts, and how you can execute R code using the SQL Server computer
- How to wrap R code in a stored procedure
- Optimizing R code for a SQL production environment
- Real-world scenarios for embedding machine learning in applications

For information about requirements and setup, see [Prerequisites](#).

R tutorials

Unless otherwise indicated, tutorials were developed for SQL Server 2016 R Services, and are expected to work in SQL Server 2017 Machine Learning Services without significant changes.

All tutorials make extensive use of features in the RevoScaleR package for SQL Server compute contexts.

- [Data Science Deep Dive with R and SQL Server](#)

Learn how to use the functions in the RevoScaleR packages. Move data between R and SQL Server, and switch compute contexts to suit a particular task. Create models and plots, and move them between your development environment and the database server.

Audience: For data scientists or developers who are already familiar with the R language, and who want to learn about the enhanced R packages and functions in Microsoft R by Revolution Analytics.

Requirements: Some basic R knowledge. Access to a server with SQL Server R Services or Machine Learning Services with R. For setup help, see [Prerequisites](#).

- [In-database R analytics for SQL developers](#)

Build and deploy a complete R solution, using only Transact-SQL tools.

Focuses on moving a solution into production. You'll learn how to wrap R code in a stored procedure, save an R model to a SQL Server database, and make parameterized calls to the R model for prediction.

Audience: For SQL developers, application developers, or SQL professionals who support R solutions and want to learn how to deploy R models to SQL Server.

Requirements: No R environment is needed. All R code is provided and you can build the complete solution using only SQL Server Management Studio and familiar business intelligence and SQL development tools. However, some basic knowledge of R is helpful.

You must have access to a SQL Server with the R language installed and enabled. For setup help, see [Prerequisites](#).

- [Quickstart: Using R in T-SQL](#)

This quickstart covers the basic syntax for using R in Transact-SQL.

Learn how to call the R run-time from T-SQL, wrap R functions in SQL code, and run a stored procedure that saves R output and R models to a SQL table.

Audience: For people who are new to the feature, and want to learn the basics of calling R from a stored procedure.

Requirements: No knowledge of R or SQL required. However, you need either SQL Server Management Studio or another client that can connect to a database and run T-SQL. We recommend the free [MSSQL extension for Visual Studio Code](#) if you are new to T-SQL queries.

You must also have access to a server with SQL Server R Services or Machine Learning Services with R already enabled. For setup help, see [Prerequisites](#).

- [Data Science End-to-End Walkthrough](#)

Demonstrates the data science process from beginning to end, as you acquire data and save it to SQL Server, analyze the data with R and build graphs.

You'll learn how to move graphics between SQL Server and R, and compare feature engineering in T-SQL with R functions. Finally, you'll learn how to use the predictive model in SQL Server for both batch scoring and single-row scoring.

Audience: For people who are familiar with R and with developer tools such as SQL Server Management Studio.

Requirements: You should have access to an R development environment and know how to run R commands. Use of PowerShell is required to download the New York City taxi dataset. You must have access to a server with SQL Server R Services or Machine Learning Services with R already enabled. For setup help, see [Prerequisites](#).

Product samples

These samples and demos are provided by the SQL Server development team to highlight the many ways that you can use embedded analytics in real-world applications.

- [Build a predictive model using R and SQL Server](#)

Learn how a ski rental business might use machine learning to predict future rentals, which helps the business plan and staff to meet future demand.

- [Perform customer clustering using R and SQL Server](#)

Use unsupervised learning to segment customers based on sales data.

Prerequisites

To use these tutorials and samples, you must install one of the following server products:

- [SQL Server 2016 R Services \(In-Database\)](#)

Supports R. Be sure to install the machine learning features, and then enable external scripting.

- [SQL Server 2017 Machine Learning Services \(In-Database\)](#)

Supports either R or Python. You must select the machine learning feature and the language to install, and then enable external scripting.

After running SQL Server setup, don't forget these important steps:

- Enable the external script execution feature by running `sp_configure 'external scripts enabled', 1`
- Restart the server
- Ensure that the service that calls the external runtime has necessary permissions
- Ensure that your SQL login or Windows user account has necessary permissions to connect to the server, to read data, and to create any database objects required by the sample

If you run into trouble, see this article for some common issues: [Troubleshooting Machine Learning Services](#)

Tutorial: Learn in-database analytics using R in SQL Server

7/16/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this tutorial for SQL programmers, you gain hands-on experience using the R language to build and deploy a machine learning solution by wrapping R code in stored procedures.

This tutorial uses a well-known public dataset, based on trips in New York city taxis. To make the sample code run quicker, we created a representative 1% sampling of the data. You'll use this data to build a binary classification model that predicts whether a particular trip is likely to get a tip or not, based on columns such as the time of day, distance, and pick-up location.

NOTE

The same solution is available in Python. SQL Server 2017 is required. See [In-database analytics for Python developers](#)

Overview

The process of building an end-to-end solution typically consists of obtaining and cleaning data, data exploration and feature engineering, model training and tuning, and finally deployment of the model in production. Development and testing of the actual code is best performed using a dedicated development environment. For R, that might mean RStudio or R Tools for Visual Studio.

However, after the solution has been created, you can easily deploy it to SQL Server using Transact-SQL stored procedures in the familiar environment of Management Studio.

- [Lesson 1: Download the sample data and scripts](#)
- [Lesson 2: Set up the tutorial environment](#)
- [Lesson 3: Explore and visualize data shape and distribution by calling R functions in stored procedures](#)
- [Lesson 4: Create data features using R in T-SQL functions](#)
- [Lesson 5: Train and save an R model using functions and stored procedures](#)
- [Lesson 6: Wrap R code in a stored procedure for operationalization](#). After the model has been saved to the database, call the model for prediction from Transact-SQL by using stored procedures.

Prerequisites

This tutorial assumes familiarity with basic database operations such as creating databases and tables, importing data, and writing SQL queries. It does not assume you know R. As such, all R code is provided. A skilled SQL programmer can use a supplied PowerShell script, sample data on GitHub, and Transact-SQL in SQL Server Management Studio to complete this example.

Before starting the tutorial:

- Verify you have a configured instance of [SQL Server 2016 R Services](#) or [SQL Server 2017 Machine Learning Services with R enabled](#). Additionally, [confirm you have the R libraries](#).

- The login that you use for this tutorial must have permissions to create databases and other objects, to upload data, select data, and run stored procedures.

NOTE

We recommend that you do **not** use SQL Server Management Studio to write or test R code. If the code that you embed in a stored procedure has any problems, the information that is returned from the stored procedure is usually inadequate to understand the cause of the error.

For debugging, we recommend you use a tool such as R Tools for Visual Studio, or RStudio. The R scripts provided in this tutorial have already been developed and debugged using traditional R tools.

Next steps

[Lesson 1: Download the sample data](#)

Lesson 1: Download data and scripts

6/8/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this step, you'll download the sample dataset and the Transact-SQL script files that are used in this tutorial. Both the data and the script files are shared on GitHub, but the PowerShell script will download the data and script files to a local directory of your choosing.

Download tutorial files from Github

1. Open a Windows PowerShell command console.

Use the option, **Run as Administrator**, if administrative privileges are needed to create the destination directory or to write files to the specified destination.

2. Run the following PowerShell commands, changing the value of the parameter *DestDir* to any local directory. The default we've used here is **TempRSQl**.

```
$source = 'https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/RSQl/Download_Scripts_SQL_Walkthrough.ps1'  
$ps1_dest = "$pwd\Download_Scripts_SQL_Walkthrough.ps1"  
$wc = New-Object System.Net.WebClient  
$wc.DownloadFile($source, $ps1_dest)  
.\\Download_Scripts_SQL_Walkthrough.ps1 -DestDir 'C:\\tempRSQl'
```

If the folder you specify in *DestDir* does not exist, it will be created by the PowerShell script.

TIP

If you get an error, you can temporarily set the policy for execution of PowerShell scripts to **unrestricted** only for this walkthrough, by using the **Bypass** argument and scoping the changes to the current session.

```
Set\\-ExecutionPolicy Bypass \\-Scope Process
```

Running this command does not result in a configuration change.

Depending on your Internet connection, the download might take a while.

3. When all files have been downloaded, the PowerShell script opens to the folder specified by *DestDir*. In the PowerShell command prompt, run the following command and review the files that have been downloaded.

```
1s
```

Results:

```
PS C:\tempRSQL> ls

Directory: C:\tempRSQL

Mode                LastWriteTime       Length Name
----                -----          ----
-a---    2/7/2016 2:19 PM           1712 create-db-tb-upload-data.sql
-a---    2/7/2016 2:19 PM           1005 fnCalculateDistance.sql
-a---    2/7/2016 2:19 PM            922 fnEngineerFeatures.sql
-a---    2/7/2016 2:19 PM        351355322 nyctaxi1pct.csv
-a---    2/7/2016 2:19 PM            821 PersistModel.sql
-a---    2/7/2016 2:19 PM           1068 PredictTipBatchMode.sql
-a---    2/7/2016 2:19 PM           1967 PredictTipSingleMode.sql
-a---    2/7/2016 2:19 PM        11751 RSQL_R_Walkthrough.R
-a---    2/7/2016 2:19 PM        13604 RunSQL_R_Walkthrough.ps1
-a---    2/7/2016 2:19 PM           3701 taxiimportfmt.xml
```

Next lesson

[Lesson 2: Import data to SQL Server using PowerShell](#)

Previous lesson

[Embedded R analytics for SQL developers](#)

Lesson 2: Set up the tutorial environment using PowerShell

6/8/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this step, you'll run a PowerShell script to create the database objects required for the walkthrough. The script creates and loads a database using sample data obtained in the previous step. It also creates functions and stored procedures used throughout the tutorial.

Create and load database objects

Among the downloaded files, you should see a PowerShell script (`RunSQL_SQL_Walkthrough.ps1`) that prepares the environment for the walkthrough. Actions performed by the script include:

- Install the SQL Native Client and SQL command-line utilities, if not already installed. These utilities are required for bulk-loading the data to the database using **bcp**.
- Create a database and tables on the SQL Server instance, and bulk-insert data sourced from a .csv file.
- Create multiple SQL functions and stored procedures.

Modify the script to use a trusted Windows identity

By default, the script assumes a SQL Server database user login and password. If you are db_owner under your Windows user account, you can use your Windows identity to create the objects. To do so, open

`RunSQL_SQL_Walkthrough.ps1` in a code editor and append `-T` to the bcp bulk insert command (line 238):

```
bcp $db_tb in $csvfilepath -t ',' -S $server -f taxiimportfmt.xml -F 2 -C "RAW" -b 200000 -U $u -P $p -T
```

Run the script to create objects

Open a PowerShell command prompt as administrator and run the following command.

```
.\RunSQL_SQL_Walkthrough.ps1
```

You are prompted to input the following information:

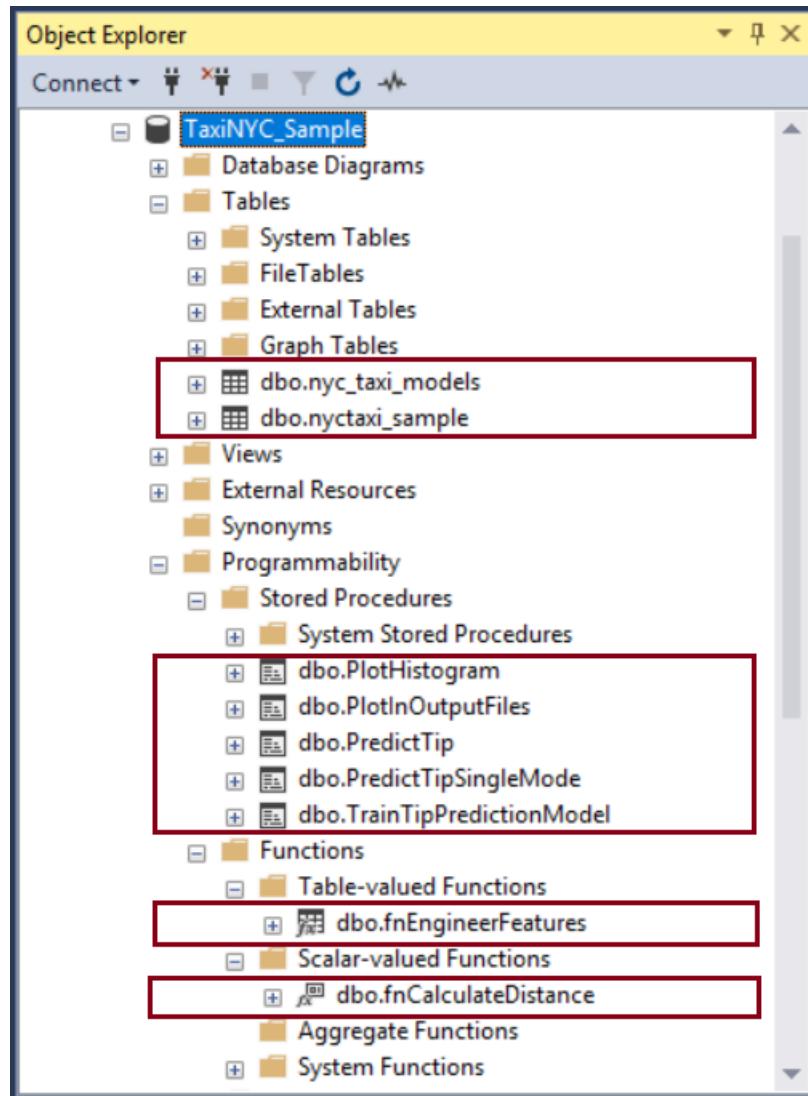
- Server instance where R Services (In-Database) has been installed. On a default instance, this can be as simple as the machine name.
- Database name. For this tutorial, scripts assume `TaxiNYC_Sample`.
- User name and user password. Enter a SQL Server database login for these values. Alternatively, if you modified the script to accept a trusted Windows identity, press Enter to leave these values blank. Your Windows identity is used on the connection.
- Fully qualified file name for the sample data downloaded in the previous lesson. For example:
`C:\tempRSQl\nyctaxi1pct.csv`

After you provide these values, the script executes immediately. During script execution, all placeholder names in

the Transact-SQL scripts are updated to use the inputs you provide.

Review database objects

When script execution is finished, confirm the database objects exist on the SQL Server instance using SQL Server Management Studio. You should see the database, tables, functions, and stored procedures.



NOTE

If the database objects already exist, they cannot be created again.

If the table already exists, the data will be appended, not overwritten. Therefore, be sure to drop any existing objects before running the script.

Objects used in this tutorial

The following table summarizes the objects created in this lesson. Although you only run one PowerShell script (`RunSQL_SQL_Walkthrough.ps1`), that script calls other SQL scripts in turn to create the objects in your database. Scripts used to create each object are mentioned in the description.

OBJECT NAME	OBJECT TYPE	DESCRIPTION
-------------	-------------	-------------

OBJECT NAME	OBJECT TYPE	DESCRIPTION
TaxiNYC_Sample	database	<p>Created by the create-db-tb-upload-data.sql script. Creates a database and two tables:</p> <p>dbo.nyctaxi_sample table: Contains the main NYC Taxi dataset. A clustered columnstore index is added to the table to improve storage and query performance. The 1% sample of the NYC Taxi dataset is inserted into this table.</p> <p>dbo.nyc_taxi_models table: Used to persist the trained advanced analytics model.</p>
fnCalculateDistance	scalar-valued function	<p>Created by the fnCalculateDistance.sql script. Calculates the direct distance between pickup and dropoff locations. This function is used in Create data features, Train and save a model and Operationalize the R model.</p>
fnEngineerFeatures	table-valued function	<p>Created by the fnEngineerFeatures.sql script. Creates new data features for model training. This function is used in Create data features and Operationalize the R model.</p>
PlotHistogram	stored procedure	<p>Created by the PlotHistogram.sql script. Calls an R function to plot the histogram of a variable and then returns the plot as a binary object. This stored procedure is used in Explore and visualize data.</p>
PlotInOutputFiles	stored procedure	<p>Created by the PlotInOutputFiles.sql script. Creates a graphic using an R function and then saves the output as a local PDF file. This stored procedure is used in Explore and visualize data.</p>
PersistModel	stored procedure	<p>Created by the PersistModel.sql script. Takes a model that has been serialized in a varbinary data type, and writes it to the specified table.</p>
PredictTip	stored procedure	<p>Created by the PredictTip.sql script. Calls the trained model to create predictions using the model. The stored procedure accepts a query as its input parameter and returns a column of numeric values containing the scores for the input rows. This stored procedure is used in Operationalize the R model.</p>

OBJECT NAME	OBJECT TYPE	DESCRIPTION
PredictTipSingleMode	stored procedure	Created by the PredictTipSingleMode.sql script. Calls the trained model to create predictions using the model. This stored procedure accepts a new observation as input, with individual feature values passed as in-line parameters, and returns a value that predicts the outcome for the new observation. This stored procedure is used in Operationalize the R model .
TrainTipPredictionModel	stored procedure	Created by the TrainTipPredictionModel.sql script. Trains a logistic regression model by calling an R package. The model predicts the value of the tipped column, and is trained using a randomly selected 70% of the data. The output of the stored procedure is the trained model, which is saved in the table nyc_taxi_models. This stored procedure is used in Train and save a model .

Next lesson

[Lesson 3: Explore and visualize the data](#)

Previous lesson

[Lesson 1: Download the sample data](#)

Lesson 3: Explore and visualize the data

6/8/2018 • 9 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this lesson, you'll review the sample data, and then generate some plots using `rxHistogram` from `RevoScaleR` and the generic `Hist` function in base R. These R functions are already included in R Services (In-Database).

A key objective is showing how to call R functions from Transact-SQL in stored procedures and save the results in application file formats:

- Create a stored procedure using **RxHistogram** to generate an R plot as varbinary data. Use **bcp** to export the binary stream to an image file.
- Create a stored procedure using **Hist** to generate a plot, saving results as JPG and PDF output.

NOTE

Because visualization is such a powerful tool for understanding data shape and distribution, R provides a range of functions and packages for generating histograms, scatter plots, box plots, and other data exploration graphs. R typically creates images using an R device for graphical output, which you can capture and store as a **varbinary** data type for rendering in application. You can also save the images to any of the support file formats (.JPG, .PDF, etc.).

Review the data

Developing a data science solution usually includes intensive data exploration and data visualization. So first take a minute to review the sample data, if you haven't already.

In the original dataset, the taxi identifiers and trip records were provided in separate files. However, to make the sample data easier to use, the two original datasets have been joined on the columns `medallion`, `hack_license`, and `pickup_datetime`. The records were also sampled to get just 1% of the original number of records. The resulting down-sampled dataset has 1,703,957 rows and 23 columns.

Taxi identifiers

- The `medallion` column represents the taxi's unique id number.
- The `hack_license` column contains the taxi driver's license number (anonymized).

Trip and fare records

- Each trip record includes the pickup and drop-off location and time, and the trip distance.
- Each fare record includes payment information such as the payment type, total amount of payment, and the tip amount.
- The last three columns can be used for various machine learning tasks. The `tip_amount` column contains continuous numeric values and can be used as the **label** column for regression analysis. The `tipped` column has only yes/no values and is used for binary classification. The `tip_class` column has multiple **class labels** and therefore can be used as the label for multi-class classification tasks.

This walkthrough demonstrates only the binary classification task; you are welcome to try building models

for the other two machine learning tasks, regression and multiclass classification.

- The values used for the label columns are all based on the *tip_amount* column, using these business rules:

DERIVED COLUMN NAME	RULE
tipped	If tip_amount > 0, tipped = 1, otherwise tipped = 0
tip_class	Class 0: tip_amount = \$0
	Class 1: tip_amount > \$0 and tip_amount <= \$5
	Class 2: tip_amount > \$5 and tip_amount <= \$10
	Class 3: tip_amount > \$10 and tip_amount <= \$20
	Class 4: tip_amount > \$20

Create a stored procedure using rxHistogram to plot the data

To create the plot, use [rxHistogram](#), one of the enhanced R functions provided in [RevoScaleR](#). This step plots a histogram based on data from a Transact-SQL query. You can wrap this function in a stored procedure, **PlotHistogram**.

- In SQL Server Management Studio, in Object Explorer, right-click the **TaxiNYC_Sample** database, expand **Programmability**, and then expand **Stored Procedures** to view the procedures created in lesson 2.
- Right-click **PlotHistogram** and select **Modify** to view the source. You can execute this procedure to call **rxHistogram** on data contained in the tipped column of nyctaxi_sample table.
- Optionally, as a learning exercise, create your own copy of this stored procedure using the following example. Open a new query window and paste in the following script to create a stored procedure that plots the histogram. This example is named **PlotHistogram2** to avoid naming collisions with the pre-existing procedure.

```
CREATE PROCEDURE [dbo].[PlotHistogram2]
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @query nvarchar(max) =
    N'SELECT tipped FROM nyctaxi_sample'
    EXECUTE sp_execute_external_script @language = N'R',
        @script = N'
    image_file = tempfile();
    jpeg(filename = image_file);
    #Plot histogram
    rxHistogram(~tipped, data=InputDataSet, col=''lightgreen'',
    title = ''Tip Histogram'', xlab = ''Tipped or not'', ylab = ''Counts'');
    dev.off();
    OutputDataSet <- data.frame(data=readBin(file(image_file, "rb"), what=raw(), n=1e6));
    ',
    @input_data_1 = @query
    WITH RESULT SETS ((plot varbinary(max)));
END
GO
```

The stored procedure **PlotHistogram2** is identical to a pre-existing stored procedure **PlotHistogram** created by the [RunSQL_SQL_Walkthrough.ps1](#) script.

- The variable `@query` defines the query text (`'SELECT tipped FROM nyctaxi_sample'`), which is passed to the R

script as the argument to the script input variable, `@input_data_1`.

- The R script is fairly simple: an R variable (`image_file`) is defined to store the image, and then the **rxHistogram** function is called to generate the plot.
- The R device is set to **off** because you are running this command as an external script in SQL Server. Typically in R, when you issue a high-level plotting command, R opens a graphics window, called a *device*. You can change the size and colors and other aspects of the window, or you can turn the device off if you are writing to a file or handling the output some other way.
- The R graphics object is serialized to an R data.frame for output.

Execute the stored procedure and use bcp to export binary data to an image file

The stored procedure returns the image as a stream of varbinary data, which obviously you cannot view directly. However, you can use the **bcp** utility to get the varbinary data and save it as an image file on a client computer.

1. In Management Studio, run the following statement:

```
EXEC [dbo].[PlotHistogram]
```

Results

`plot 0xFFD8FFE000104A4649...`

2. Open a PowerShell command prompt and run the following command, providing the appropriate instance name, database name, username, and credentials as arguments. For those using Windows identities, you can replace **-U** and **-P** with **-T**.

```
bcp "exec PlotHistogram" queryout "plot.jpg" -S <SQL Server instance name> -d TaxiNYC_Sample -U <user name> -P <password>
```

NOTE

Command switches for bcp are case-sensitive.

3. If the connection is successful, you will be prompted to enter more information about the graphic file format. Press ENTER at each prompt to accept the defaults, except for these changes:

- For **prefix-length of field plot**, type 0
- Type **Y** if you want to save the output parameters for later reuse.

```
Enter the file storage type of field plot [varbinary(max)]:  
Enter prefix-length of field plot [8]: 0  
Enter length of field plot [0]:  
Enter field terminator [none]:
```

```
Do you want to save this format information in a file? [Y/n]  
Host filename [bcp.fmt]:
```

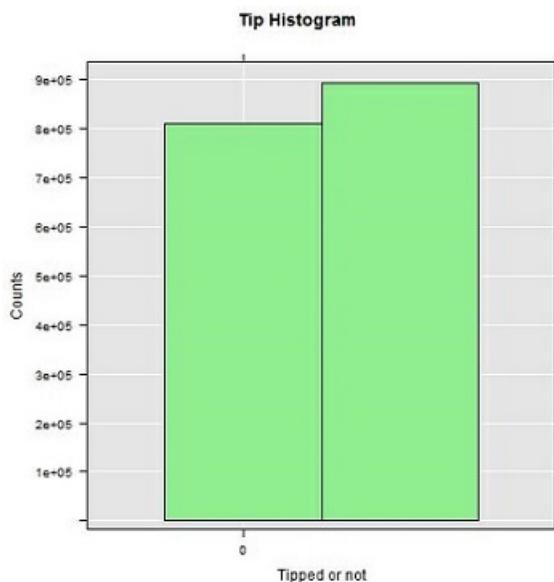
Results

```
Starting copy...
1 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total      : 3922    Average : (0.25 rows per sec.)
```

TIP

If you save the format information to file (bcp.fmt), the **bcp** utility generates a format definition that you can apply to similar commands in future without being prompted for graphic file format options. To use the format file, add `-f bcp.fmt` to the end of any command line, after the password argument.

4. The output file will be created in the same directory where you ran the PowerShell command. To view the plot, just open the file plot.jpg.



Create a stored procedure using Hist and multiple output formats

Typically, data scientists generate multiple data visualizations to get insights into the data from different perspectives. In this example, the stored procedure uses the Hist function to create the histogram, exporting the binary data to popular formats such as .JPG, .PDF, and .PNG.

1. Use the existing stored procedure, **PlotInOutputFiles**, to write histograms, scatterplots, and other R graphics to .JPG and .PDF format. The `RunSQL_SQL_Walkthrough.ps1` creates **PlotInOutputFiles** and adds it to the database. Use right-click **Modify** to view the source.
2. Optionally, as a learning exercise, create your own copy of the procedure as **PlotInOutputFiles2**, with a unique name to avoid a naming conflict.

In SQL Server Management Studio, open a new **Query** window, and paste in the following Transact-SQL statement.

```

CREATE PROCEDURE [dbo].[PlotInOutputFiles2]
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @query nvarchar(max) =
    N'SELECT cast(tipped as int) as tipped, tip_amount, fare_amount FROM [dbo].[nyctaxi_sample]'
    EXECUTE sp_execute_external_script @language = N'R',
    @script = N'
        # Set output directory for files and check for existing files with same names
        mainDir <- "C:\\temp\\plots"
        dir.create(mainDir, recursive = TRUE, showWarnings = FALSE)
        setwd(mainDir);
        print("Creating output plot files:", quote=FALSE)

        # Open a jpeg file and output histogram of tipped variable in that file.
        dest_filename = tempfile(pattern = "rHistogram_Tipped_", tmpdir = mainDir)
        dest_filename = paste(dest_filename, ".jpg", sep="")
        print(dest_filename, quote=FALSE);
        jpeg(filename=dest_filename);
        hist(InputDataSet$tipped, col = "lightgreen", xlab="Tipped",
              ylab = "Counts", main = "Histogram, Tipped");
        dev.off();

        # Open a pdf file and output histograms of tip amount and fare amount.
        # Outputs two plots in one row
        dest_filename = tempfile(pattern = "rHistograms_Tip_and_Fare_Amount_", tmpdir = mainDir)
        dest_filename = paste(dest_filename, ".pdf", sep="")
        print(dest_filename, quote=FALSE);
        pdf(file=dest_filename, height=4, width=7);
        par(mfrow=c(1,2));
        hist(InputDataSet$tip_amount, col = "lightgreen",
              xlab="Tip amount ($)",
              ylab = "Counts",
              main = "Histogram, Tip amount", xlim = c(0,40), 100);
        hist(InputDataSet$fare_amount, col = "lightgreen",
              xlab="Fare amount ($)",
              ylab = "Counts",
              main = "Histogram,
              Fare amount",
              xlim = c(0,100), 100);
        dev.off();

        # Open a pdf file and output an xyplot of tip amount vs. fare amount using lattice;
        # Only 10,000 sampled observations are plotted here, otherwise file is large.
        dest_filename = tempfile(pattern = "rXYPlots_Tip_vs_Fare_Amount_", tmpdir = mainDir)
        dest_filename = paste(dest_filename, ".pdf", sep="")
        print(dest_filename, quote=FALSE);
        pdf(file=dest_filename, height=4, width=4);
        plot(tip_amount ~ fare_amount,
              data = InputDataSet[sample(nrow(InputDataSet), 10000), ],
              ylim = c(0,50),
              xlim = c(0,150),
              cex=.5,
              pch=19,
              col='darkgreen',
              main = "Tip amount by Fare amount",
              xlab="Fare Amount ($)",
              ylab = "Tip Amount ($)");
        dev.off();'
    @input_data_1 = @query
END

```

- The output of the SELECT query within the stored procedure is stored in the default R data frame, `InputDataSet`. Various R plotting functions can then be called to generate the actual graphics files. Most of the embedded R script represents options for these graphics functions, such as `plot` or `hist`.

- All files are saved to the local folder `C:\temp\Plots`. The destination folder is defined by the arguments provided to the R script as part of the stored procedure. You can change the destination folder by changing the value of the variable, `mainDir`.
- To output the files to a different folder, change the value of the `mainDir` variable in the R script embedded in the stored procedure. You can also modify the script to output different formats, more files, and so on.

Execute the stored procedure

Run the following statement to export binary plot data to JPEG and PDF file formats.

```
EXEC PlotInOutFiles
```

Results

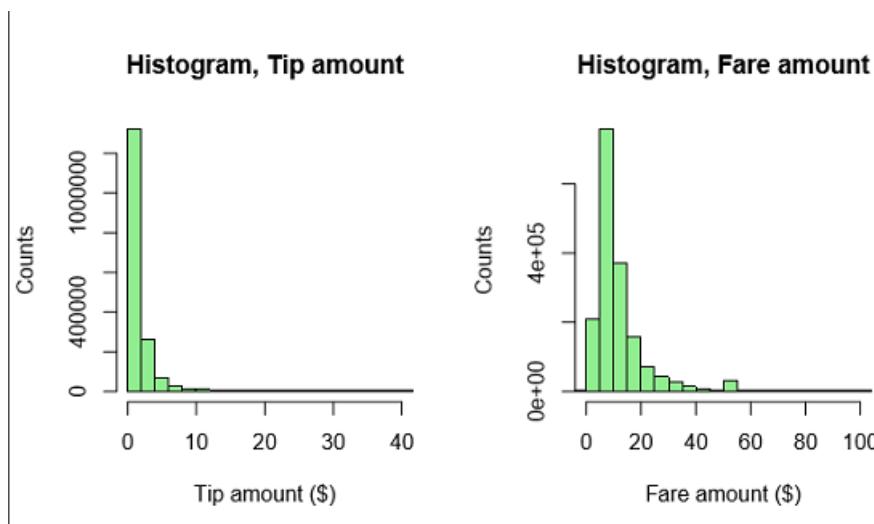
```
STDOUT message(s) from external script:  
[1] Creating output plot files:[1] C:\temp\plots\rHistogram_Tipped_18887f6265d4.jpg[1]  
  
C:\temp\plots\rHistograms_Tip_and_Fare_Amount_1888441e542c.pdf[1]  
  
C:\temp\plots\rXYPlots_Tip_vs_Fare_Amount_18887c9d517b.pdf
```

The numbers in the file names are randomly generated to ensure that you don't get an error when trying to write to an existing file.

View output

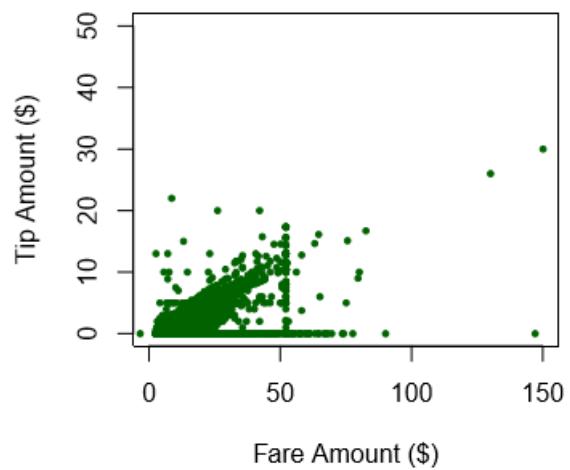
To view the plot, open the destination folder and review the files that were created by the R code in the stored procedure.

1. Go the folder indicated in the STDOUT message (in the example, this is `C:\temp\plots`)
2. Open `rHistogram_Tipped.jpg` to show the number of trips that got a tip vs. the trips that got no tip. (This histogram is much like the one you generated in the previous step.)
3. Open `rHistograms_Tip_and_Fare_Amount.pdf` to view distribution of tip amounts, plotted against the fare amounts.



4. Open `rXYPlots_Tip_vs_Fare_Amount.pdf` to view a scatterplot with the fare amount on the x-axis and the tip amount on the y-axis.

Tip amount by Fare amount



Next lesson

[Lesson 4: Create data features using T-SQL](#)

Previous lesson

[Lesson 2: Prepare the tutorial environment using PowerShell](#)

Lesson 4: Create data features using R and T-SQL

6/8/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this step, you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

About feature engineering

After several rounds of data exploration, you have collected some insights from the data, and are ready to move on to *feature engineering*. This process of creating meaningful features from the raw data is a critical step in creating analytical models.

In this dataset, the distance values are based on the reported meter distance, and don't necessarily represent geographical distance or the actual distance traveled. Therefore, you'll need to calculate the direct distance between the pick-up and drop-off points, by using the coordinates available in the source NYC Taxi dataset. You can do this by using the [Haversine formula](#) in a custom Transact-SQL function.

You'll use one custom T-SQL function, *fnCalculateDistance*, to compute the distance using the Haversine formula, and use a second custom T-SQL function, *fnEngineerFeatures*, to create a table containing all the features.

The overall process is as follows:

- Create the T-SQL function that performs the calculations
- Call the function to generate the feature data
- Save the feature data to a table

Calculate trip distance using *fnCalculateDistance*

The function *fnCalculateDistance* should have been downloaded and registered with SQL Server as part of the preparation for this tutorial. Take a minute to review the code.

1. In Management Studio, expand **Programmability**, expand **Functions** and then **Scalar-valued functions**.
2. Right-click *fnCalculateDistance*, and select **Modify** to open the Transact-SQL script in a new query window.

```

CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
-- User-defined function that calculates the direct distance between two geographical coordinates.
RETURNS float
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2) * COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
    BEGIN
        SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) / @distance);
    END
    RETURN @distance
END
GO

```

- The function is a scalar-valued function, returning a single data value of a predefined type.
- It takes latitude and longitude values as inputs, obtained from trip pick-up and drop-off locations. The Haversine formula converts locations to radians and uses those values to compute the direct distance in miles between those two locations.

Generate the features using *fnEngineerFeatures*

To add the computed values to a table that can be used for training the model, you'll use another function, *fnEngineerFeatures*. The new function calls the previously created T-SQL function, *fnCalculateDistance*, to get the direct distance between pick-up and drop-off locations.

1. Take a minute to review the code for the custom T-SQL function, *fnEngineerFeatures*, which should have been created for you as part of the preparation for this walkthrough.

```

CREATE FUNCTION [dbo].[fnEngineerFeatures] (
    @passenger_count int = 0,
    @trip_distance float = 0,
    @trip_time_in_secs int = 0,
    @pickup_latitude float = 0,
    @pickup_longitude float = 0,
    @dropoff_latitude float = 0,
    @dropoff_longitude float = 0)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT
        @passenger_count AS passenger_count,
        @trip_distance AS trip_distance,
        @trip_time_in_secs AS trip_time_in_secs,
        [dbo].[fnCalculateDistance](@pickup_latitude, @pickup_longitude, @dropoff_latitude,
        @dropoff_longitude) AS direct_distance
)
GO

```

- This table-valued function that takes multiple columns as inputs, and outputs a table with multiple feature columns.

- The purpose of this function is to create new features for use in building a model.
2. To verify that this function works, use it to calculate the geographical distance for those trips where the metered distance was 0 but the pick-up and drop-off locations were different.

```
SELECT tipped, fare_amount, passenger_count,(trip_time_in_secs/60) as TripMinutes,  
trip_distance, pickup_datetime, dropoff_datetime,  
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS  
direct_distance  
FROM nyctaxi_sample  
WHERE pickup_longitude != dropoff_longitude and pickup_latitude != dropoff_latitude and  
trip_distance = 0  
ORDER BY trip_time_in_secs DESC
```

As you can see, the distance reported by the meter doesn't always correspond to geographical distance. This is why feature engineering is so important. You can use these improved data features to train a machine learning model using R.

Next lesson

[Lesson 5: Train and save a model using T-SQL](#)

Previous lesson

[Lesson 3: Explore and visualize the data using R and stored procedures](#)

Lesson 5: Train and save a model using T-SQL

6/8/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this lesson, you'll learn how to train a machine learning model by using R. You'll train the model using the data features you created in the previous lesson, and then save the trained model in a SQL Server table. In this case, the R packages are already installed with R Services (In-Database), so everything can be done from SQL.

Create the stored procedure

When calling R from T-SQL, you use the system stored procedure, `sp_execute_external_script`. However, for processes that you repeat often, such as retraining a model, it is easier to encapsulate the call to `sp_execute_external_script` in another stored procedure.

1. First, create a stored procedure that contains the R code to build the tip prediction model. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure `TrainTipPredictionModel`. This stored procedure defines the input data and uses an R package to create a logistic regression model.

```
CREATE PROCEDURE [dbo].[TrainTipPredictionModel]
AS
BEGIN
    DECLARE @inquery nvarchar(max) = N'
        select tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
        pickup_datetime, dropoff_datetime,
        dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as direct_distance
        from nyctaxi_sample
        tablesample (70 percent) repeatable (98052)
    '

    -- Insert the trained model into a database table
    INSERT INTO nyc_taxi_models
    EXEC sp_execute_external_script @language = N'R',
                                    @script = N'

## Create model
logitObj <- rxLogit(tipped ~ passenger_count + trip_distance + trip_time_in_secs + direct_distance,
data = InputDataSet)
summary(logitObj)

## Serialize model and put it in data frame
trained_model <- data.frame(model=as.raw(serializerserialize(logitObj, NULL)));
', @input_data_1 = @inquery,
@output_data_1_name = N'trained_model'
;

END
GO
```

- However, to ensure that some data is left over to test the model, 70% of the data are randomly selected from the taxi data table.

- The SELECT query uses the custom scalar function `fnCalculateDistance` to calculate the direct distance between the pick-up and drop-off locations. the results of the query are stored in the default R input variable, `InputDataset`.
 - The R script calls the `rxLogit` function, which is one of the enhanced R functions included with R Services (In-Database), to create the logistic regression model.
- The binary variable `tipped` is used as the *label* or outcome column, and the model is fit using these feature columns: `passenger_count`, `trip_distance`, `trip_time_in_secs`, and `direct_distance`.
- The trained model, saved in the R variable `logitObj`, is serialized and put in a data frame for output to SQL Server. That output is inserted into the database table `nyc_taxi_models`, so that you can use it for future predictions.

2. Run the statement to create the stored procedure, if it doesn't already exist.

Generate the R model using the stored procedure

Because the stored procedure already includes a definition of the input data, you don't need to provide an input query.

1. To generate the R model, call the stored procedure without any other parameters:

```
EXEC TrainTipPredictionModel
```

2. Watch the **Messages** window of Management Studio for messages that would be piped to R's `stdout` stream, like this message:

"STDOUT message(s) from external script: Rows Read: 1193025, Total Rows Processed: 1193025, Total Chunk Time: 0.093 seconds"

You might also see messages specific to the individual function, `rxLogit`, displaying the variables and test metrics generated as part of model creation.

3. When the statement has completed, open the table `nyc_taxi_models`. Processing of the data and fitting the model might take a while.

You can see that one new row has been added, which contains the serialized model in the column `model`.

```
model
-----
0x580A000000002000302020....
```

In the next step you'll use the trained model to create predictions.

Next lesson

[Lesson 6: Operationalize the model](#)

Previous lesson

[Lesson 4: Create data features using R and T-SQL functions](#)

Lesson 6: Predict potential outcomes using an R model in a stored procedure

6/8/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial for SQL developers on how to use R in SQL Server.

In this step, you learn to use the model against new observations to predict potential outcomes. The model is wrapped in a stored procedure which can be called directly by other applications. The walkthrough demonstrates several ways to perform scoring:

- **Batch scoring mode:** Use a SELECT query as an input to the stored procedure. The stored procedure returns a table of observations corresponding to the input cases.
- **Individual scoring mode:** Pass a set of individual parameter values as input. The stored procedure returns a single row or value.

First, let's see how scoring works in general.

Basic scoring

The stored procedure **PredictTip** illustrates the basic syntax for wrapping a prediction call in a stored procedure.

```
CREATE PROCEDURE [dbo].[PredictTip] @inquery nvarchar(max)
AS
BEGIN

DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM nyc_taxi_models);
EXEC sp_execute_external_script @language = N'R',
    @script = N'
    mod <- unserialize(as.raw(model));
    print(summary(mod))
    OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL, predVarNames = "Score",
    type = "response", writeModelVars = FALSE, overwrite = TRUE);
    str(OutputDataSet)
    print(OutputDataSet)
    ',
    @input_data_1 = @inquery,
    @params = N'@model varbinary(max)',
    @model = @lmodel2
    WITH RESULT SETS ((Score float));
END
GO
```

- The SELECT statement gets the serialized model from the database, and stores the model in the R variable `mod` for further processing using R.
- The new cases for scoring are obtained from the Transact-SQL query specified in `@inquery`, the first parameter to the stored procedure. As the query data is read, the rows are saved in the default data frame, `InputDataSet`. This data frame is passed to the `rxPredict` function in `RevoScaleR`, which generates the scores.

```
OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL, predVarNames =
"Score", type = "response", writeModelVars = FALSE, overwrite = TRUE);
```

Because a data.frame can contain a single row, you can use the same code for batch or single scoring.

- The value returned by the `rxPredict` function is a **float** that represents the probability that the driver gets a tip of any amount.

Batch scoring

Now let's see how batch scoring works.

1. Let's start by getting a smaller set of input data to work with. This query creates a "top 10" list of trips with passenger count and other features needed to make a prediction.

```
SELECT TOP 10 a.passenger_count AS passenger_count, a.trip_time_in_secs AS trip_time_in_secs,
a.trip_distance AS trip_distance, a.dropoff_datetime AS dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude,dropoff_longitude) AS
direct_distance

FROM (SELECT medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime, pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude FROM nyctaxi_sample)a

LEFT OUTER JOIN

(SELECT medallion, hack_license, pickup_datetime FROM nyctaxi_sample TABLESAMPLE (70 percent)
REPEATABLE (98052)      )b

ON a.medallion=b.medallion AND a.hack_license=b.hack_license
AND a.pickup_datetime=b.pickup_datetime
WHERE b.medallion IS NULL
```

Sample results

	passenger_count	trip_time_in_secs	trip_distance	dropoff_datetime	direct_distance
1	283	0.7	2013-03-27 14:54:50.000	0.5427964547	
1	289	0.7	2013-02-24 12:55:29.000	0.3797099614	
1	214	0.7	2013-06-26 13:28:10.000	0.6970098661	

This query can be used as input to the stored procedure, **PredictTipMode**, provided as part of the download.

2. Take a minute to review the code of the stored procedure **PredictTipMode** in Management Studio.

```

***** Object:  StoredProcedure [dbo].[PredictTipMode] *****/
CREATE PROCEDURE [dbo].[PredictTipMode] @inquery nvarchar(max)
AS
BEGIN
DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM nyc_taxi_models);
EXEC sp_execute_external_script
    @language = N'R',
    @script = N'
    mod <- unserialize(as.raw(model));
    print(summary(mod))
    OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL, predVarNames =
"Score", type = "response", writeModelVars = FALSE, overwrite = TRUE);
    str(OutputDataSet)
    print(OutputDataSet)
    ',
    @input_data_1 = @inquery,
    @params = N'@model varbinary(max)',
    @model = @lmodel2
    WITH RESULT SETS ((Score float));
END

```

- Provide the query text in a variable and pass it as a parameter to the stored procedure:

```

-- Define the input data
DECLARE @query_string nvarchar(max)
SET @query_string='SELECT TOP 10 a.passenger_count as passenger_count, a.trip_time_in_secs AS
trip_time_in_secs, a.trip_distance AS trip_distance, a.dropoff_datetime AS dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude,dropoff_longitude) AS
direct_distance FROM  (SELECT medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime, pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude FROM nyctaxi_sample )a  LEFT OUTER JOIN (SELECT medallion,
hack_license, pickup_datetime FROM nyctaxi_sample TABLESAMPLE (70 percent) REPEATABLE (98052))b ON
a.medallion=b.medallion AND a.hack_license=b.hack_license AND a.pickup_datetime=b.pickup_datetime
WHERE b.medallion is null'

-- Call the stored procedure for scoring and pass the input data
EXEC [dbo].[PredictTip] @inquery = @query_string;

```

- The stored procedure returns a series of values representing the prediction for each of the top 10 trips. However, the top trips are also single-passenger trips with a relatively short trip distance, for which the driver is unlikely to get a tip.

TIP

Rather than returning just the "yes-tip" and "no-tip" results, you could also return the probability score for the prediction, and then apply a WHERE clause to the *Score* column values to categorize the score as "likely to tip" or "unlikely to tip", using a threshold value such as 0.5 or 0.7. This step is not included in the stored procedure but it would be easy to implement.

Single-row scoring

Sometimes you want to pass in individual values from an application and get a single result based on those values. For example, you could set up an Excel worksheet, web application, or Reporting Services report to call the stored procedure and provide inputs typed or selected by users.

In this section, you learn how to create single predictions using a stored procedure.

- Take a minute to review the code of the stored procedure **PredictTipSingleMode**, which is included as part of the download.

```

CREATE PROCEDURE [dbo].[PredictTipSingleMode] @passenger_count int = 0, @trip_distance float = 0,
@trip_time_in_secs int = 0, @pickup_latitude float = 0, @pickup_longitude float = 0, @dropoff_latitude
float = 0, @dropoff_longitude float = 0
AS
BEGIN
DECLARE @inquery nvarchar(max) = N'SELECT * FROM [dbo].[fnEngineerFeatures](@passenger_count,
@trip_distance, @trip_time_in_secs, @pickup_latitude, @pickup_longitude, @dropoff_latitude,
@dropoff_longitude)';
DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM nyc_taxi_models);
EXEC sp_execute_external_script
    @language = N'R',
    @script = N'
    mod <- unserialize(as.raw(model));
    print(summary(mod));
    OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL, predVarNames =
"Score", type = "response", writeModelVars = FALSE, overwrite = TRUE);
    str(OutputDataSet);
    print(OutputDataSet);
    ',
    @input_data_1 = @inquery,
    @params = N'@model varbinary(max),@passenger_count int,@trip_distance float,@trip_time_in_secs int ,
@pickup_latitude float ,@pickup_longitude float ,@dropoff_latitude float ,@dropoff_longitude float',
    @model = @lmodel2, @passenger_count =@passenger_count, @trip_distance=@trip_distance,
    @trip_time_in_secs=@trip_time_in_secs, @pickup_latitude=@pickup_latitude,
    @pickup_longitude=@pickup_longitude, @dropoff_latitude=@dropoff_latitude,
    @dropoff_longitude=@dropoff_longitude
    WITH RESULT SETS ((Score float));
END

```

- This stored procedure takes multiple single values as input, such as passenger count, trip distance, and so forth.

If you call the stored procedure from an external application, make sure that the data matches the requirements of the R model. This might include ensuring that the input data can be cast or converted to an R data type, or validating data type and data length.

- The stored procedure creates a score based on the stored R model.

2. Try it out, by providing the values manually.

Open a new **Query** window, and call the stored procedure, providing values for each of the parameters. The parameters represent feature columns used by the model and are required.

```

EXEC [dbo].[PredictTipSingleMode] @passenger_count = 0,
@trip_distance = 2.5,
@trip_time_in_secs = 631,
@pickup_latitude = 40.763958,
@pickup_longitude = -73.973373,
@dropoff_latitude = 40.782139,
@dropoff_longitude = 73.977303

```

Or, use this shorter form supported for [parameters to a stored procedure](#):

```
EXEC [dbo].[PredictTipSingleMode] 1, 2.5, 631, 40.763958,-73.973373, 40.782139,-73.977303
```

3. The results indicate that the probability of getting a tip is low on these top 10 trips, since all are single-passenger trips over a relatively short distance.

Conclusions

This concludes the tutorial. Now that you have learned to embed R code in stored procedures, you can extend these practices to build models of your own. The integration with Transact-SQL makes it much easier to deploy R models for prediction and to incorporate model retraining as part of an enterprise data workflow.

Previous lesson

[Lesson 5: Train and save an R model using T-SQL](#)

End-to-end data science walkthrough for R and SQL Server

7/16/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this walkthrough, you develop an end-to-end solution for predictive modeling based on R feature support in either SQL Server 2016 or SQL Server 2017.

This walkthrough is based on a popular set of public data, the New York City taxi dataset. You use a combination of R code, SQL Server data, and custom SQL functions to build a classification model that indicates the probability that the driver might get a tip on a particular taxi trip. You also deploy your R model to SQL Server and use server data to generate scores based on the model.

This example can be extended to all kinds of real-life problems, such as predicting customer responses to sales campaigns, or predicting spending or attendance at events. Because the model can be invoked from a stored procedure, you can easily embed it in an application.

Because the walkthrough is designed to introduce R developers to R Services (In-Database), R is used wherever possible. However, this does not mean that R is necessarily the best tool for each task. In many cases, SQL Server might provide better performance, particularly for tasks such as data aggregation and feature engineering. Such tasks can particularly benefit from new features in SQL Server 2017, such as memory optimized columnstore indexes. We try to point out possible optimizations along the way.

Target audience

This walkthrough is intended for R or SQL developers. It provides an introduction into how R can be integrated into enterprise workflows using R Services (In-Database). You should be familiar with database operations, such as creating databases and tables, importing data, and running queries.

- All SQL and R scripts are included.
- You might need to modify strings in the scripts, to run in your environment. You can do this with any code editor, such as [Visual Studio Code](#).

Prerequisites

We recommend that you do this walkthrough on a laptop or other computer that has the Microsoft R libraries installed. You must be able to connect, on the same network, to a SQL Server computer with SQL Server and the R language enabled.

You can run the walkthrough on a computer that has both SQL Server and an R development environment but we don't recommend this configuration for a production environment.

If you want to run R commands from a remote computer, such as a laptop or other networked computer, you must install the Microsoft R Open libraries. You can install either Microsoft R Client or Microsoft R Server. The remote computer must be able to connect to the SQL Server instance.

If you need to put client and server on the same computer, be sure to install a separate set of Microsoft R libraries for use in sending R script from a "remote" client. Do not use the R libraries that are installed for use by the SQL Server instance for this purpose.

Add R to SQL Server

You must have access to an instance of SQL Server with support for R installed. This walkthrough was originally developed for SQL Server 2016 and tested on 2017, so you should be able to use either of the following SQL Server versions. (There are some small differences in the RevoScaleR functions between the releases.)

- [Install SQL Server 2017 Machine Learning Services](#)
- [Install SQL Server 2016 R Services.](#)

Install an R development environment

For this walkthrough, we recommend that you use an R development environment. Here are some suggestions:

- **R Tools for Visual Studio** (RTVS) is a free plug-in that provides Intellisense, debugging, and support for Microsoft R. You can use it with both R Server and SQL Server Machine Learning Services. To download, see [R Tools for Visual Studio](#).
- **Microsoft R Client** is a lightweight development tool that supports development in R using the RevoScaleR package. To get it, see [Get Started with Microsoft R Client](#).
- **RStudio** is one of the more popular environments for R development. For more information, see <https://www.rstudio.com/products/RStudio/>.

You cannot complete this tutorial using a generic installation of RStudio or other environment; you must also install the R packages and connectivity libraries for Microsoft R Open. For more information, see [Set Up a Data Science Client](#).

- Basic R tools (R.exe, RTerm.exe, RScripts.exe) are also installed by default when you install R in SQL Server or R Client. If you do not wish to install an IDE, you can use these tools.

Get permissions on the SQL Server instance and database

To connect to an instance of SQL Server to run scripts and upload data, you must have a valid login on the database server. You can use either a SQL login or integrated Windows authentication. Ask the database administrator to configure the following permissions for the account, in the database where you use R:

- Create database, tables, functions, and stored procedures
- Write data into tables
- Ability to run R script (`GRANT EXECUTE ANY EXTERNAL SCRIPT to <user>`)

For this walkthrough, we have used the SQL login **RTestUser**. We generally recommend that you use Windows integrated authentication, but using the SQL login is simpler for some demo purposes.

Next steps

[Prepare the data using PowerShell](#)

Prepare the data using PowerShell (walkthrough)

7/16/2018 • 12 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

By this time, you should have one of the following installed:

- SQL Server 2016 R Services
- SQL Server 2017 Machine Learning Services, with the R language enabled

In this lesson, you download the data, R packages, and R scripts used in the walkthrough from a Github repository. You can download everything using a PowerShell script for convenience.

You also need to install some additional R packages, both on the server and on your R workstation. The steps are described.

Then, you use a second PowerShell script, RunSQL_R_Walkthrough.ps1, to configure the database that is used for modeling and scoring. That script performs a bulk load of the data into the database you specify, and then creates some SQL functions and stored procedures that simplify data science tasks.

Let's get started!

1. Download the data and scripts

All the code needed has been provided in a GitHub repository. You can use a PowerShell script to make a local copy of the files.

1. On your data science client computer, open a Windows PowerShell command prompt as administrator.
2. To ensure that you can run the download script without an error, run this command. It temporarily allows scripts without changing system defaults.

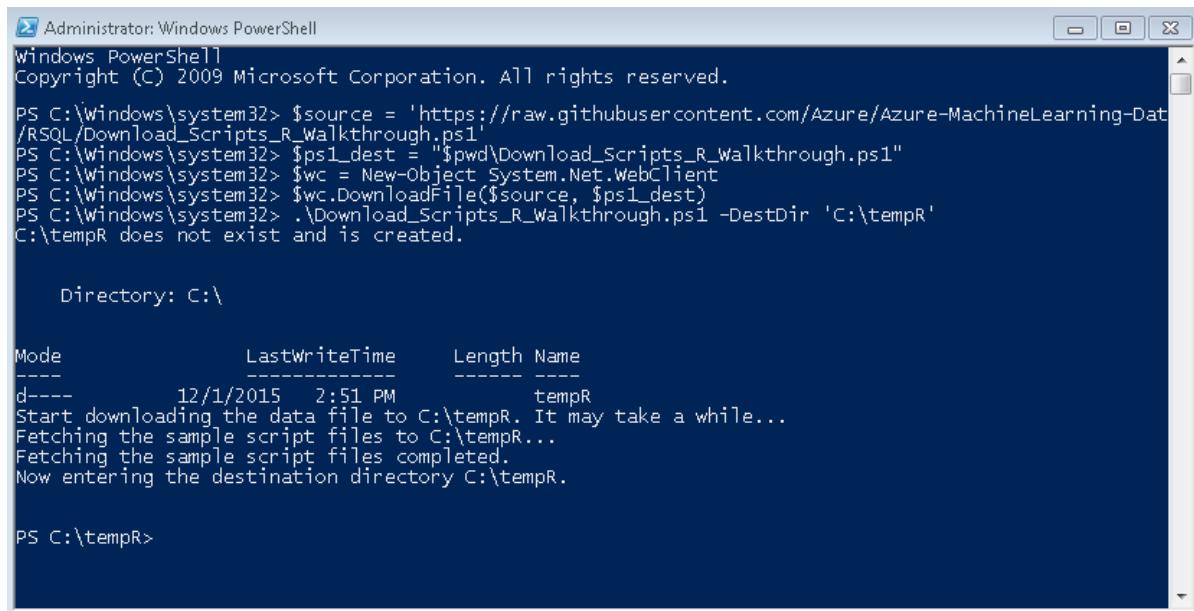
```
Set-ExecutionPolicy Unrestricted -Scope Process -Force
```

3. Run the following Powershell command to download script files to a local directory. If you do not specify a different directory, by default the folder `C:\tempR` is created and all files saved there.

```
$source = 'https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/RSQL/Download_Scripts_R_Walkthrough.ps1'  
$ps1_dest = "$pwd\Download_Scripts_R_Walkthrough.ps1"  
$wc = New-Object System.Net.WebClient  
$wc.DownloadFile($source, $ps1_dest)  
.\\Download_Scripts_R_Walkthrough.ps1 -DestDir 'C:\tempR'
```

If you want to save the files in a different directory, edit the values of the parameter `DestDir` and specify a different folder on your computer. If you type a folder name that does not exist, the PowerShell script creates the folder for you.

4. Downloading might take a while. After it is complete, the Windows PowerShell command console should look like this:



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> $source = 'https://raw.githubusercontent.com/Azure/Azure-MachineLearning-Dat
/RSQl/Download_Scripts_R_Walkthrough.ps1'
PS C:\Windows\system32> $ps1_dest = "$pwd\Download_Scripts_R_Walkthrough.ps1"
PS C:\Windows\system32> $wc = New-Object System.Net.WebClient
PS C:\Windows\system32> $wc.DownloadFile($source, $ps1_dest)
PS C:\Windows\system32> .\Download_Scripts_R_Walkthrough.ps1 -DestDir 'C:\tempR'
C:\tempR does not exist and is created.

    Directory: C:\  

Mode                LastWriteTime     Length Name
----                -----          ---- 
d----       12/1/2015  2:51 PM           tempR
Start downloading the data file to C:\tempR. It may take a while...
Fetching the sample script files to C:\tempR...
Fetching the sample script files completed.
Now entering the destination directory C:\tempR.

PS C:\tempR>
```

5. In the PowerShell console, you can run the command `ls` to view a list of the files that were downloaded to `DestDir`. For a description of the files, see [What's Included](#).

2. Install required R packages

This walkthrough requires some R libraries that are not installed by default as part of R Services (In-Database). You must install the packages both on the client where you develop the solution, and on the SQL Server computer where you deploy the solution.

Install required packages on the client

The R script that you downloaded includes the commands to download and install these packages.

1. In your R environment, open the script file, RSQL_R_Walkthrough.R.
2. Highlight and execute these lines.

```
# Install required R libraries, if they are not already installed.
if (!('ggmap' %in% rownames(installed.packages()))){install.packages('ggmap')}
if (!('mapproj' %in% rownames(installed.packages()))){install.packages('mapproj')}
if (!('ROCR' %in% rownames(installed.packages()))){install.packages('ROCR')}
if (!('RODBC' %in% rownames(installed.packages()))){install.packages('RODBC')}
```

Some packages also install required packages. In all, about 32 packages are required.

Install required packages on the server

There are many different ways that you can install packages on SQL Server. For example, SQL Server provides [R package management](#) feature that lets database administrators create a package repository and assign user the rights to install their own packages. However, if you are an administrator on the computer, you can install new packages using R, as long as you install to the correct library.

NOTE

On the server, **do not** install to a user library even if prompted. If you install to a user library, the SQL Server instance cannot find or run the packages. For more information, see [Installing New R Packages on SQL Server](#).

1. On the SQL Server computer, open RGui.exe **as an administrator**. If you have installed SQL Server R Services using the defaults, RGui.exe can be found in C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\bin\x64).

2. At an R prompt, run the following R commands:

```
install.packages("ggmap", lib=grep("Program Files", .libPaths(), value=TRUE)[1])
install.packages("mapproj", lib=grep("Program Files", .libPaths(), value=TRUE)[1])
install.packages("ROCR", lib=grep("Program Files", .libPaths(), value=TRUE)[1])
install.packages("RODBC", lib=grep("Program Files", .libPaths(), value=TRUE)[1])
```

- This example uses the R grep function to search the vector of available paths and find the path that includes "Program Files". For more information, see <http://www.rdocumentation.org/packages/base/functions/grep>.
- If you think the packages are already installed, check the list of installed packages by running `installed.packages()`.

3. Prepare the environment using RunSQL_R_Walkthrough.ps1

Along with the data files, R scripts, and T-SQL scripts, the download includes the PowerShell script

`RunSQL_R_Walkthrough.ps1`. The script performs these actions:

- Checks whether the SQL Native Client and command-line utilities for SQL Server are installed. The command-line tools are needed to run the [bcp Utility](#), which is used for fast bulk loading of data into SQL tables.
- Connects to the specified instance of SQL Server and runs some Transact-SQL scripts that configure the database and create the tables for the model and data.
- Runs a SQL script to create several stored procedures.
- Loads the data you downloaded previously into a table named `nyctaxi_sample`.
- Rewrites the arguments in the R script file to use the database name that you specify.

Run this script on the computer where you build the solution: for example, the laptop where you develop and test your R code. This computer, which we'll call the data science client, must be able to connect to the SQL Server computer using the Named Pipes protocol.

1. Open a PowerShell command line **as administrator**.

2. Navigate to the folder where you downloaded the scripts, and type the name of the script as shown. Press ENTER.

```
.\RunSQL_R_Walkthrough.ps1
```

3. You are prompted for each of the following parameters:

Database server name: The name of the SQL Server instance where Machine learning Services or R Services is installed.

Depending on your network's requirements, the instance name might require qualification with one or more subnet names. For example, if MY SERVER doesn't work, try myserver.subnet.mycompany.com.

Name of the database you want to create: For example, you might type **Tutorial** or **Taxi**

User name: Provide an account that has databases access privileges. There are two options:

- Type the name of a SQL login that has CREATE DATABASE privileges, and provide the SQL password on a successive prompt.
- Press ENTER without typing any name to use your own Windows identity, and at the secured prompt,

type your Windows password. PowerShell does not support entering a different Windows user name.

- If you fail to specify a valid user, the script defaults to using integrated Windows authentication.

WARNING

When you use the prompt in the PowerShell script to provide your credentials, the password is written to the updated script file in plain text. Edit the file to remove the credentials immediately after you have created the necessary R objects.

Path to the csv file: Provide the full path to the data file. The default path and filename is

C:\tempR\nyctaxi1pct.csv .

4. Press ENTER to run the script.

The script should download the file and load the data into the database automatically. This can take a while. Watch the status messages in the PowerShell window.

If bulk import or any other step fails, you can load the data manually as described in the [Troubleshooting](#) section.

Results (successful completion)

```
Execution successful
Completed registering all stored procedures used in this walkthrough.
This step (registering all stored procedures) takes 0.39 seconds.
Plug in the database server name, database name, user name and password into the R script file
This step (plugging in database information) takes 0.48 seconds.
```

Click this link to jump to the next lesson: [View and explore the data using SQL](#)

Troubleshooting

If you have any problems with the PowerShell script, you can run all or any of the steps manually, using the lines of the PowerShell script as examples. This section lists some common issues and workarounds.

PowerShell script didn't download the data

To download the data manually, right-click the following link and select **Save target as**.

<http://getgoing.blob.core.windows.net/public/nyctaxi1pct.csv>

Make a note of the path to the downloaded data file and the file name where the data was saved. You need the full path to load the data to the table using **bcp**.

Unable to download the data

The data file is large. You must use a computer that has a relatively good Internet connection, or the download might time out.

Could not connect or script failed

You might get this error when running one of the scripts: *A network-related or instance-specific error has occurred while establishing a connection to SQL Server*

- Check the spelling of your instance name.
- Verify the complete connection string.
- Depending on your network's requirements, the instance name might require qualification with one or more subnet names. For example, if MYSERVER doesn't work, try myserver.subnet.mycompany.com.
- Check whether Windows Firewall allows connections by SQL Server.

- Try registering your server and making sure that it allows remote connections.
- If you are using a named instance, enable SQL Browser to make connections easier.

Network error or protocol not found

- Verify that the instance supports remote connections.
- Verify that the specified SQL user can connect remotely to the database.
- Enable Named Pipes on the instance.
- Check permissions for the account. The account that you specified might not have the permissions to create a new database and upload data.

bcp did not run

- Verify that the **bcp Utility** is available on your computer. You can run **bcp** from either a PowerShell window or a Windows command prompt.
- If you get an error, add the location of the **bcp** utility to the PATH system environment variable and try again.

The table schema was created but the table has no data

If the rest of the script ran without problems, you can upload the data to the table manually by calling **bcp** from the command line as follows:

Using a SQL login

```
bcp TutorialDB.dbo.nyctaxi_sample in c:\tempR\nyctaxi1pct.csv -t ',' -S rtestserver.contoso.com -f
C:\tempR\taxiimportfmt.xml -F 2 -C "RAW" -b 200000 -U <SQL login> -P <password>
```

Using Windows authentication

```
bcp TutorialDB.dbo.nyctaxi_sample in c:\tempR\nyctaxi1pct.csv -t ',' -S rtestserver.contoso.com -f
C:\tempR\taxiimportfmt.xml -F 2 -C "RAW" -b 200000 -T
```

- The **in** keyword specifies the direction of data movement.
- The **-f** argument requires that you specify the full path of a format file. A format file is required if you use the **in** option.
- Use the **-U** and **-P** arguments if running bcp with a SQL login.
- Use the **-T** argument if you are using Windows integrated authentication.

If the script doesn't load the data, check the syntax, and verify that your server name is specified correctly for your network. For example, be sure to include any subnets, and include the computer name if you are connecting to a named instance. Verify that the login has the ability to perform bulk uploads.

I want to run the script without prompts

You can specify all the parameters in a single command line, using this template, with values specific to your environment.

```
.\RunSQL_R_Walkthrough.ps1 -server <server address> -dbname <new db name> -u <user name> -p <password> -
 csvfilepath <path to csv file>
```

The following example runs the script using a SQL login:

```
.\RunSQL_R_Walkthrough.ps1 -server MyServer.subnet.domain.com -dbname MyDB -u SqlUserName -p SqlUserPassword
 -csvfilepath C:\temp\nyctaxi1pct.csv
```

- Connects to the specified instance and database using the credentials of *SqlUserName*.
- Gets data from the file *C:\temp\nyctaxi1pct.csv*.

- Loads the data into the table `dbo.nyctaxi_sample`, in the database `MyDB` on the SQL Server instance named `MyServer`.

The data loaded but it contains duplicates

If your database contains an existing table of the same name and the same schema, `bcp` inserts a new copy of the data rather than overwriting existing data.

To avoid duplicate data, truncate any existing tables before running the script again.

What's included in the sample

When you download the files from the GitHub repository, you get the following:

- Data in CSV format; see [Training and scoring data](#) for details
- A PowerShell script for preparing the environment
- An XML format file for importing the data to SQL Server using bcp
- Multiple T-SQL scripts
- All the R code you need to run this walkthrough

Training and scoring data

The data is a representative sampling of the New York City taxi data set, which contains records of over 173 million individual trips in 2013, including the fares and tip amounts paid for each trip. To make the data easier to work with, the Microsoft data science team performed downsampling to get just 1% of the data. This data has been shared in a public blob storage container in Azure, in .CSV format. The source data is an uncompressed file, just under 350 MB.

- Public dataset: [NYC Taxi and Limousine Commission](#)
- [Building Azure ML Models on the NYC Taxi Dataset]
(<https://blogs.technet.microsoft.com/machinelearning/2015/04/02/building-azure-ml-models-on-the-nyc-taxi-dataset/>).

PowerShell and R script files

- **RunSQL_R_Walkthrough.ps1** You run this script first, using PowerShell. It calls the SQL scripts to load data into the database.
- **taxiimportfmt.xml** A format definition file that is used by the BCP utility to load data into the database.
- **RSQL_R_Walkthrough.R** This is the core R script that is used in rest of the lessons for doing your data analysis and modeling. It provides all the R code that you need to explore SQL Server data, build the classification model, and create plots.

T-SQL script files

The PowerShell script executes multiple Transact-SQL scripts on the SQL Server instance. The following table lists the Transact-SQL scripts and what they do.

SQL SCRIPT FILE NAME	DESCRIPTION
create-db-tb-upload-data.sql	<p>Creates database and two tables:</p> <p><i>nyctaxi_sample</i>: Table that stores the training data, a one-percent sample of the NYC taxi data set. A clustered columnstore index is added to the table to improve storage and query performance.</p> <p><i>nyc_taxi_models</i>: A table used for storing trained models in binary format.</p>

SQL SCRIPT FILE NAME	DESCRIPTION
PredictTipBatchMode.sql	Creates a stored procedure that calls a trained model to predict the labels for new observations. It accepts a query as its input parameter.
PredictTipSingleMode.sql	Creates a stored procedure that calls a trained classification model to predict the labels for new observations. Variables of the new observations are passed in as in-line parameters.
PersistModel.sql	Creates a stored procedure that helps store the binary representation of the classification model in a table in the database.
fnCalculateDistance.sql	Creates a SQL scalar-valued function that calculates the direct distance between pick-up and drop-off locations.
fnEngineerFeatures.sql	Creates a SQL table-valued function that creates features for training the classification model

The T-SQL queries used in this walkthrough have been tested and can be run as-is in your R code. However, if you want to experiment further or develop your own solution, we recommend that you use a dedicated SQL development environment to test and tune your queries first, before adding them to your R code.

- The [mssql extension](#) for [Visual Studio Code](#) is a free, lightweight environment for running queries that also supports most database development tasks.
- [SQL Server Management Studio](#) is a powerful but free tool provided for development and management of SQL Server databases.

Next lesson

[View and explore the data using R and SQL](#)

Previous lesson

[End-to-end data science walkthrough for R and SQL Server](#)

View and explore the data using SQL (walkthrough)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Data exploration is an important part of modeling data, and involves reviewing summaries of data objects to be used in the analyses, as well as data visualization. In this lesson, you explore the data objects and generate plots, using both Transact-SQL and R functions included in R Services (In-Database).

Then you generate plots to visualize the data, using new functions provided by packages installed with R Services (In-Database).

TIP

Already an R maestro?

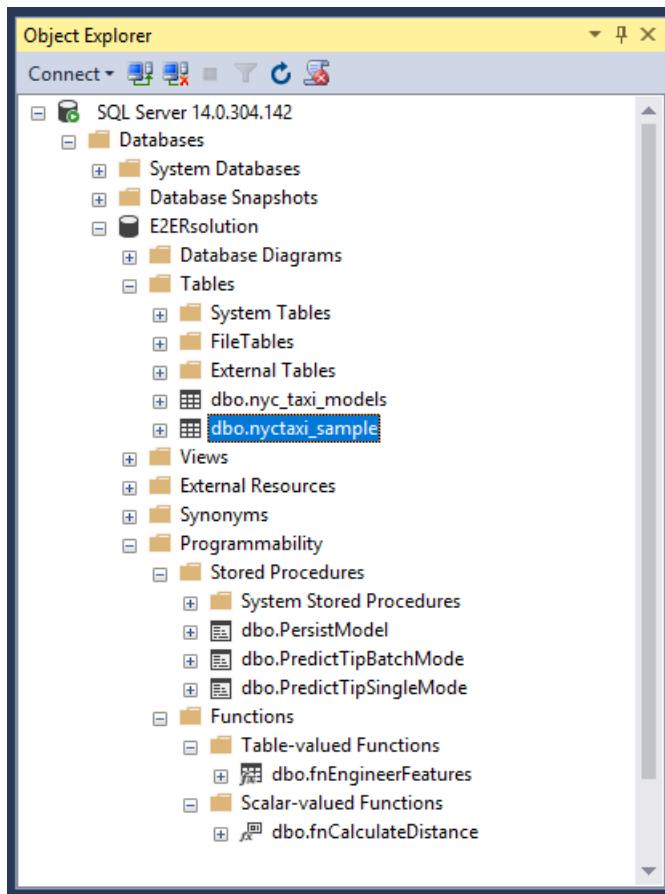
Now that you've downloaded all the data and prepared the environment, you are welcome to run the complete R script in RStudio or any other environment, and explore the functionality on your own. Just open the file `RSQL_Walkthrough.R` and highlight and run individual lines, or run the entire script as a demo.

To get additional explanations of the RevoScaleR functions, and tips for working with SQL Server data in R, continue with the tutorial. It uses exactly the same script.

Verify downloaded data using SQL Server

First, take a minute to ascertain that your data was loaded correctly.

1. Connect to your SQL Server instance using your favorite database management tool, such as SQL Server, Server Explorer in Visual Studio, or Visual Studio Code.
2. Select the database you created, and expand to see the new database, tables, and functions.



3. To verify that the data loaded correctly, right-click the table and select **Select TOP 1000 rows**. The menu option runs this query:

```
SELECT TOP 1000 * FROM [dbo].[nyctaxi_sample]
```

If you don't see any data in the table, refer to the [Troubleshooting](#) section in the previous topic.

4. This data table has been optimized for set-based calculations, by adding a [columnstore index](#). Run this statement to generate a quick summary on the table.

```
SELECT DISTINCT [passenger_count]
    , ROUND (SUM ([fare_amount]),0) as TotalFares
    , ROUND (AVG ([fare_amount]),0) as AvgFares
FROM [dbo].[nyctaxi_sample]
GROUP BY [passenger_count]
ORDER BY AvgFares DESC
```

In the next lesson, you'll generate some more complex summaries using R.

Next lesson

[Summarize data using R](#)

Previous lesson

[Prepare the data using PowerShell](#)

View and summarize data using R

4/11/2018 • 8 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Now let's work with the same data using R code. In this lesson, you learn how to use the functions in the **RevoScaleR** package.

An R script is provided with this walkthrough that includes all the code needed to create the data object, generate summaries, and build models. The R script file, **RSQL_RWalkthrough.R**, can be found in the location where you installed the script files.

- If you are experienced with R, you can run the script all at once.
- For people learning to use RevoScaleR, this tutorial goes through the script line by line.
- To run individual lines from the script, you can highlight a line or lines in the file and press Ctrl + ENTER.

TIP

Save your R workspace in case you want to complete the rest of the walkthrough later. That way the data objects and other variables are ready for re-use.

Define a SQL Server compute context

Microsoft R makes it easy to get data from SQL Server to use in your R code. The process is as follows:

- Create a connection to a SQL Server instance
- Define a query that has the data you need, or specify a table or view
- Define one or more compute contexts to use when running R code
- Optionally, define transformations that are applied to the data source while it is being read from the source

The following steps are all part of the R code and should be run in an R environment. We used Microsoft R Client, because it includes all the RevoScaleR packages, as well as a basic, lightweight set of R tools.

1. If the **RevoScaleR** package is not already loaded, run this line of R code:

```
library("RevoScaleR")
```

The quotation marks are optional, in this case, though recommended.

If you get an error, make sure that your R development environment is using a library that includes the RevoScaleR package. Use a command such as `.libPaths()` to view the current library path.

2. Create the connection string for SQL Server and save it in an R variable, `connStr`.

```
connStr <- "Driver=SQL  
Server;Server=your_server_name;Database=Your_Database_Name;Uid=Your_User_Name;Pwd=Your_Password"
```

For the server name, you might be able to use only the instance name, or you might need to fully qualify the name, depending on your network.

For Windows authentication, the syntax is a bit different:

```
connStr <- "Driver=SQL Server;Server=SQL_instance_name;Database=database_name;Trusted_Connection=Yes"
```

The R script available for download uses SQL logins only. Generally, we recommend that you use Windows authentication where possible, to avoid saving passwords in your R code. However, to ensure that the code in this tutorial matches the code downloaded from Github, we'll use a SQL login for the rest of the walkthrough.

3. Define variables to use in creating a new *compute context*. After you create the compute context object, you can use it to run R code on the SQL Server instance.

```
sqlShareDir <- paste("C:\\AllShare\\\", Sys.getenv("USERNAME"), sep = "")  
sqlWait <- TRUE  
sqlConsoleOutput <- FALSE
```

- R uses a temporary directory when serializing R objects back and forth between your workstation and the SQL Server computer. You can specify the local directory that is used as *sqlShareDir*, or accept the default.
- Use *sqlWait* to indicate whether you want R to wait for results from the server. For a discussion of waiting vs. non-waiting jobs, see [Distributed and parallel computing with ScaleR in Microsoft R](#).
- Use the argument *sqlConsoleOutput* to indicate that you don't want to see output from the R console.

4. You call the [RxInSqlServer](#) constructor to create the compute context object with the variables and connection strings already defined, and save the new object in the R variable *sqlcc*.

```
sqlcc <- RxInSqlServer(connectionString = connStr, shareDir = sqlShareDir, wait = sqlWait,  
consoleOutput = sqlConsoleOutput)
```

5. By default, the compute context is local, so you need to explicitly set the *active* compute context.

```
rxSetComputeContext(sqlcc)
```

- `rxSetComputeContext` returns the previously active compute context invisibly so that you can use it
- `rxGetComputeContext` returns the active compute context

Note that setting a compute context only affects operations that use functions in the **RevoScaleR** package; the compute context does not affect the way that open source R operations are performed.

Create a data source using RxSqlServer

In Microsoft R, a *data source* is an object you create using RevoScaleR functions. The data source object specifies some set of data that you want to use for a task, such as model training or feature extraction. You can get data from a variety of sources; for the list of currently supported sources, see [RxDataSource](#).

Earlier you defined a connection string, and saved that information in an R variable. You can re-use that connection information to specify the data you want to get.

1. Save a SQL query as a string variable. The query defines the data for training the model.

```
sampleDataQuery <- "SELECT tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
pickup_datetime, dropoff_datetime, pickup_longitude, pickup_latitude, dropoff_longitude,
dropoff_latitude FROM nyctaxi_sample"
```

We've used a TOP clause here to make things run faster, but the actual rows returned by the query can vary depending on order. Hence, your summary results might also be different from those listed below. Feel free to remove the TOP clause.

2. Pass the query definition as an argument to the [RxSqlServerData](#) function.

```
inDataSource <- RxSqlServerData(
  sqlQuery = sampleDataQuery,
  connectionString = connStr,
  colClasses = c(pickup_longitude = "numeric", pickup_latitude = "numeric",
  dropoff_longitude = "numeric", dropoff_latitude = "numeric"),
  rowsPerRead=500
)
```

- The argument *colClasses* specifies the column types to use when moving the data between SQL Server and R. This is important because SQL Server uses different data types than R, and more data types. For more information, see [R Libraries and Data Types](#).
 - The argument *rowsPerRead* is important for managing memory usage and efficient computations. Most of the enhanced analytical functions in R Services (In-Database) process data in chunks and accumulate intermediate results, returning the final computations after all of the data has been read. By adding the *rowsPerRead* parameter, you can control how many rows of data are read into each chunk for processing. If the value of this parameter is too large, data access might be slow because you don't have enough memory to efficiently process such a large chunk of data. On some systems, setting *rowsPerRead* to an excessively small value can also provide slower performance.
3. At this point, you've created the *inDataSource* object, but it doesn't contain any data. The data is not pulled from the SQL query into the local environment until you run a function such as [rxImport](#) or [rxSummary](#).

However, now that you've defined the data objects, you can use it as the argument to other functions.

Use the SQL Server data in R summaries

In this section, you'll try out several of the functions provided in R Services (In-Database) that support remote compute contexts. By applying R functions to the data source, you can explore, summarize, and chart the SQL Server data.

1. Call the function [rxGetVarInfo](#) to get a list of the variables in the data source and their data types.

rxGetVarInfo is a handy function; you can call it on any data frame, or on a set of data in a remote data object, to get information such as the maximum and minimum values, the data type, and the number of levels in factor columns.

Consider running this function after any kind of data input, feature transformation, or feature engineering. By doing so, you can ensure that all the features you want to use in your model are of the expected data type and avoid errors.

```
rxGetVarInfo(data = inDataSource)
```

Results

```
Var 1: tipped, Type: integer
Var 2: fare_amount, Type: numeric
Var 3: passenger_count, Type: integer
Var 4: trip_time_in_secs, Type: numeric, Storage: int64
Var 5: trip_distance, Type: numeric
Var 6: pickup_datetime, Type: character
Var 7: dropoff_datetime, Type: character
Var 8: pickup_longitude, Type: numeric
Var 9: pickup_latitude, Type: numeric
Var 10: dropoff_longitude, Type: numeric
```

2. Now, call the RevoScaleR function `rxSummary` to get more detailed statistics about individual variables.

`rxSummary` is based on the R `summary` function, but has some additional features and advantages. `rxSummary` works in multiple compute contexts and supports chunking. You can also use `rxSummary` to transform values, or summarize based on factor levels.

In this example, you summarize the fare amount based on the number of passengers.

```
start.time <- proc.time()
rxSummary(~fare_amount:F(passenger_count,1,6), data = inDataSource)
used.time <- proc.time() - start.time
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2)," seconds,
Elapsed Time=", round(used.time[3],2),
" seconds to summarize the inDataSource.", sep=""))
```

- The first argument to `rxSummary` specifies the formula or term to summarize by. Here, the `F()` function is used to convert the values in *passenger_count* into factors before summarizing. You also have to specify the minimum value (1) and maximum value (6) for the *passenger_count* factor variable.
- If you do not specify the statistics to output, by default `rxSummary` outputs Mean, StDev, Min, Max, and the number of valid and missing observations.
- This example also includes some code to track the time the function starts and completes, so that you can compare performance.

Results

If the `rxSummary` function runs successfully, you should see results like these, followed by a list of statistics by category.

```
rxSummary(formula = ~fare_amount:F(passenger_count, 1,6), data = inDataSource)
Data: inDataSource (RxSqlServerData Data Source)
Number of valid observations: 1000
```

Bonus exercise on big data

Try defining a new query string with all the rows. we recommend you set up a new data source object for this experiment. You might also try changing the `rowsToRead` parameter to see how it affects throughput.

```

bigDataQuery <- "SELECT tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
pickup_datetime, dropoff_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude FROM
nyctaxi_sample"

bigDataSource <- RxSqlServerData(
  sqlQuery = bigDataQuery,
  connectionString = connStr,
  colClasses = c(pickup_longitude = "numeric", pickup_latitude = "numeric",
dropoff_longitude = "numeric", dropoff_latitude = "numeric"),
  rowsPerRead=500
)

start.time <- proc.time()
rxSummary(~fare_amount:F(passenger_count,1,6), data = bigDataSource)
used.time <- proc.time() - start.time
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2)," seconds,
Elapsed Time=", round(used.time[3],2),
" seconds to summarize the inDataSource.", sep=""))

```

TIP

While this is running, you can use a tool like [Process Explorer](#) or SQL Profiler to see how the connection is made and the R code is run using SQL Server services.

Another option is to monitor R jobs running on SQL Server using these [custom reports](#).

Next lesson

[Create graphs and plots using R](#)

Previous lesson

[Explore the data using SQL](#)

Create graphs and plots using SQL and R (walkthrough)

5/30/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

In this part of the walkthrough, you learn techniques for generating plots and maps using R with SQL Server data. You create a simple histogram, to get some practice, and then develop a more complex map plot.

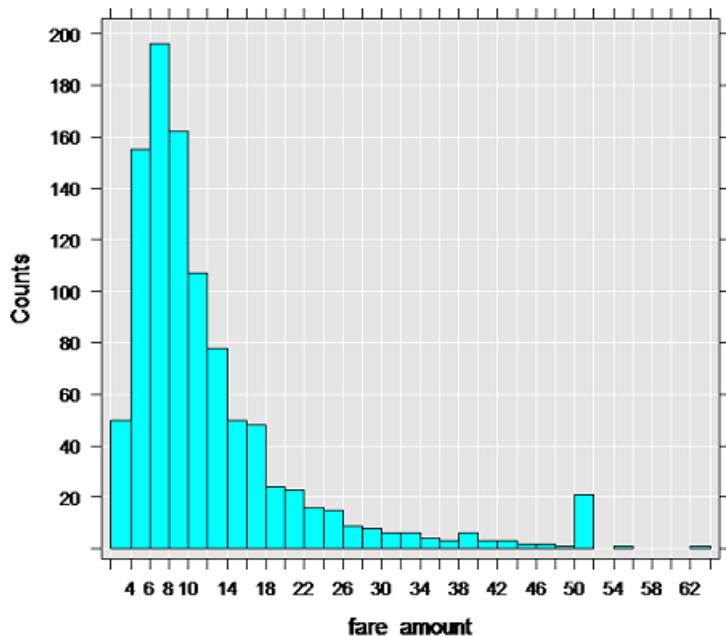
Create a histogram

1. Generate the first plot, using the [rxHistogram](#) function. The rxHistogram function provides functionality similar to that in open source R packages, but can run in a remote execution context.

```
# Plot fare amount on SQL Server and return the plot
start.time <- proc.time()
rxHistogram(~fare_amount, data = inDataSource, title = "Fare Amount Histogram")
used.time <- proc.time() - start.time
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2), " seconds, Elapsed Time=",
round(used.time[3],2), " seconds to generate plot.", sep=""))
```

2. The image is returned in the R graphics device for your development environment. For example, in RStudio, click the **Plot** window. In R Tools for Visual Studio, a separate graphics window is opened.

Fare Amount Histogram



NOTE

Does your graph look different?

That's because `inDataSource` uses only the top 1000 rows. The ordering of rows using `TOP` is non-deterministic in the absence of an `ORDER BY` clause, so it's expected that the data and the resulting graph might vary. This particular image was generated using about 10,000 rows of data. We recommend that you experiment with different numbers of rows to get different graphs, and note how long it takes to return the results in your environment.

Create a map plot

Typically, database servers block Internet access. This can be inconvenient when using R packages that need to download maps or other images to generate plots. However, there is a workaround that you might find useful when developing your own applications. Basically, you generate the map representation on the client, and then overlay on the map the points that are stored as attributes in the SQL Server table.

1. Define the function that creates the R plot object. The custom function `mapPlot` creates a scatter plot that uses the taxi pickup locations, and plots the number of rides that started from each location. It uses the **ggplot2** and **ggmap** packages, which should already be installed and loaded.

```
mapPlot <- function(inDataSource, googMap){  
  library(ggmap)  
  library(mapproj)  
  ds <- rxImport(inDataSource)  
  p <- ggmap(googMap)+  
    geom_point(aes(x = pickup_longitude, y =pickup_latitude ), data=ds, alpha =.5,  
    color="darkred", size = 1.5)  
  return(list(myplot=p))  
}
```

- The `mapPlot` function takes two arguments: an existing data object, which you defined earlier using `RxSqlServerData`, and the map representation passed from the client.
 - In the line beginning with the `ds` variable, `rxImport` is used to load into memory data from the previously created data source, `inDataSource`. (That data source contains only 1000 rows; if you want to create a map with more data points, you can substitute a different data source.)
 - Whenever you use **open source** R functions, data must be loaded into data frames in local memory. However, by calling the `rxImport` function, you can run in the memory of the remote compute context.
2. Change the compute context to local, and load the libraries required for creating the maps.

```
rxSetComputeContext("local")  
library(ggmap)  
library(mapproj)  
gc <- geocode("Times Square", source = "google")  
googMap <- get_googlemap(center = as.numeric(gc), zoom = 12, maptype = 'roadmap', color = 'color');
```

- The `gc` variable stores a set of coordinates for Times Square, NY.
 - The line beginning with `googmap` generates a map with the specified coordinates at the center.
3. Switch to the SQL Server compute context, and render the results, by wrapping the plot function in `rxExec` as shown here. The `rxExec` function is part of the **RevoScaleR** package, and supports execution of arbitrary R functions in a remote compute context.

```

rxSetComputeContext(sqlcc)
myplots <- rxExec(mapPlot, inDataSource, googMap, timesToRun = 1)
plot(myplots[[1]][["myplot"]]);

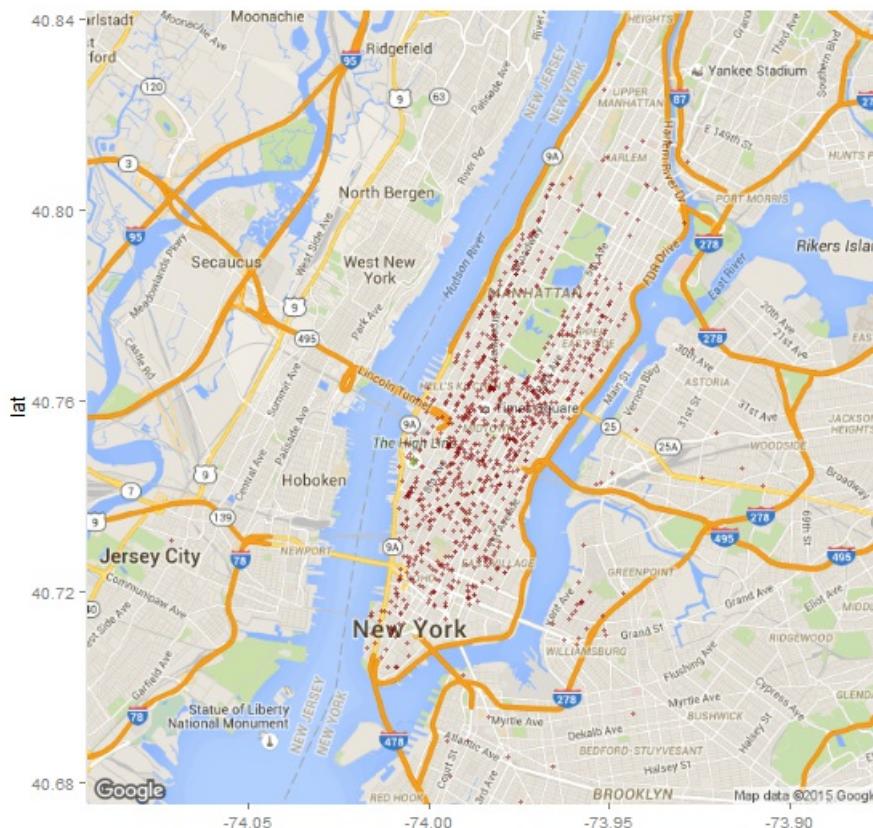
```

- The map data in `googMap` is passed as an argument to the remotely executed function `mapPlot`. Because the maps were generated in your local environment, they must be passed to the function in order to create the plot in the context of SQL Server.
- When the line beginning with `plot` runs, the rendered data is serialized back to the local R environment so that you can view it in your R client.

NOTE

If you're using SQL Server in an Azure virtual machine, you might get an error at this point. That's because a default firewall rule in Azure blocks network access by R code. For details on how to fix this error, see [Installing R Services in an Azure VM](#).

- The following image shows the output plot. The taxi pickup locations are added to the map as red dots. Your image might look different, depending how many locations are in the data source you used.



Next lesson

[Create data features using R and SQL](#)

Previous lesson

[Summarize data using R](#)

Create data features using R and SQL (walkthrough)

4/11/2018 • 8 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Data engineering is an important part of machine learning. Data often requires transformation before you can use it for predictive modeling. If the data does not have the features you need, you can engineer them from existing values.

For this modeling task, rather than using the raw latitude and longitude values of the pickup and drop-off location, you'd like to have the distance in miles between the two locations. To create this feature, you compute the direct linear distance between two points, by using the [haversine formula](#).

In this step, we compare two different methods for creating a feature from data:

- Using a custom R function
- Using a custom T-SQL function in Transact-SQL

The goal is to create a new SQL Server set of data that includes the original columns plus the new numeric feature, *direct_distance*.

Featurization using R

The R language is well-known for its rich and varied statistical libraries, but you still might need to create custom data transformations.

First, let's do it the way R users are accustomed to: get the data onto your laptop, and then run a custom R function, *ComputeDist*, which calculates the linear distance between two points specified by latitude and longitude values.

1. Remember that the data source object you created earlier gets only the top 1000 rows. So let's define a query that gets all the data.

```
bigQuery <- "SELECT tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,  
pickup_datetime, dropoff_datetime, pickup_latitude, pickup_longitude, dropoff_latitude,  
dropoff_longitude FROM nyctaxi_sample";
```

2. Create a new SQL Server data source using the query.

```
featureDataSource <- RxSqlServerData(sqlQuery = bigQuery, colClasses = c(pickup_longitude = "numeric",  
pickup_latitude = "numeric", dropoff_longitude = "numeric", dropoff_latitude = "numeric",  
passenger_count = "numeric", trip_distance = "numeric", trip_time_in_secs = "numeric",  
direct_distance = "numeric"), connectionString = connStr);
```

- [RxSqlServerData](#) can take either a query consisting of a valid SELECT query, provided as the argument to the *sqlQuery* parameter, or the name of a table object, provided as the *table* parameter.
 - If you want to sample data from a table, you must use the *sqlQuery* parameter, define sampling parameters using the T-SQL TABLESAMPLE clause, and set the *rowBuffering* argument to FALSE.
3. Run the following code to create the custom R function. *ComputeDist* takes in two pairs of latitude and longitude values, and calculates the linear distance between them, returning the distance in miles.

```

env <- new.env();
env$ComputeDist <- function(pickup_long, pickup_lat, dropoff_long, dropoff_lat){
  R <- 6371/1.609344 #radius in mile
  delta_lat <- dropoff_lat - pickup_lat
  delta_long <- dropoff_long - pickup_long
  degrees_to_radians = pi/180.0
  a1 <- sin(delta_lat/2*degrees_to_radians)
  a2 <- as.numeric(a1)^2
  a3 <- cos(pickup_lat*degrees_to_radians)
  a4 <- cos(dropoff_lat*degrees_to_radians)
  a5 <- sin(delta_long/2*degrees_to_radians)
  a6 <- as.numeric(a5)^2
  a <- a2+a3*a4*a6
  c <- 2*atan2(sqrt(a),sqrt(1-a))
  d <- R*c
  return (d)
}

```

- The first line defines a new environment. In R, an environment can be used to encapsulate name spaces in packages and such. You can use the `search()` function to view the environments in your workspace. To view the objects in a specific environment, type `ls(<envname>)`.
 - The lines beginning with `$env.ComputeDistance` contain the code that defines the haversine formula, which calculates the *great-circle distance* between two points on a sphere.
4. Having defined the function, you apply it to the data to create a new feature column, *direct_distance*. but before you run the transformation, change the compute context to local.

```
rxSetComputeContext("local");
```

5. Call the `rxDataStep` function to get the feature engineering data, and apply the `env$ComputeDist` function to the data in memory.

```

start.time <- proc.time();

changed_ds <- rxDataStep(inData = featureEngineeringQuery,
transforms = list(direct_distance=ComputeDist(pickup_longitude,pickup_latitude, dropoff_longitude,
dropoff_latitude),
tipped = "tipped", fare_amount = "fare_amount", passenger_count = "passenger_count",
trip_time_in_secs = "trip_time_in_secs", trip_distance="trip_distance",
pickup_datetime = "pickup_datetime", dropoff_datetime = "dropoff_datetime"),
transformEnvir = env,
rowsPerRead=500,
reportProgress = 3);

used.time <- proc.time() - start.time;
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2)," seconds, Elapsed Time=",
round(used.time[3],2), " seconds to generate features.", sep=""));

```

- The `rxDataStep` function supports various methods for modifying data in place. For more information, see this article: [How to transform and subset data in Microsoft R](#)

However, a couple of points worth noting regarding `rxDataStep`:

In other data sources, you can use the arguments `varsToKeep` and `varsToDelete`, but these are not supported for SQL Server data sources. Therefore, in this example, we've used the `transforms` argument to specify both the pass-through columns and the transformed columns. Also, when running in a SQL Server compute context, the `inData` argument can only take a SQL Server data source.

The preceding code can also produce a warning message when run on larger data sets. When the number of rows times the number of columns being created exceeds a set value (the default is 3,000,000), rxDataStep returns a warning, and the number of rows in the returned data frame will be truncated. To remove the warning, you can modify the *maxRowsByCols* argument in the rxDataStep function. However, if *maxRowsByCols* is too large, you might experience problems when loading the data frame into memory.

6. Optionally, you can call [rxGetVarInfo](#) to inspect the schema of the transformed data source.

```
rxGetVarInfo(data = changed_ds);
```

Featurization using Transact-SQL

Now, create a custom SQL function, *ComputeDist*, to accomplish the same task as the custom R function.

1. Define a new custom SQL function, named *fnCalculateDistance*. The code for this user-defined SQL function is provided as part of the PowerShell script you ran to create and configure the database. The function should already exist in your database.

If it does not exist, use SQL Server Management Studio to generate the function in the same database where the taxi data is stored.

```
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
-- User-defined function calculates the direct distance between two geographical coordinates.
RETURNS
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2) * COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
    BEGIN
        SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) / @distance);
    END
    RETURN @distance
END
```

2. Run the following Transact-SQL statement from any application that supports Transact-SQL, just to see how the function works.

```
SELECT tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance, pickup_datetime,
dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
direct_distance,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample
```

3. Having defined this function, it would be easy to create the features you want by using SQL and then insert the values directly into a new table:

```

SELECT tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance, pickup_datetime,
dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
direct_distance,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
INTO NewFeatureTable
FROM nyctaxi_sample

```

4. However, let's see how to call the custom SQL function from R code. First, store the SQL featurization query in an R variable.

```

featureEngineeringQuery = "SELECT tipped, fare_amount, passenger_count,
    trip_time_in_secs, trip_distance, pickup_datetime, dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
    direct_distance,
    pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
    FROM nyctaxi_sample
    tablesample (1 percent) repeatable (98052)"

```

TIP

This query has been modified to get a smaller sample of data, to make this walkthrough faster. You can remove the TABLESAMPLE clause if you want to get all the data; however, depending on your environment, it might not be possible to load the full dataset into R, resulting in an error.

5. Use the following lines of code to call the Transact-SQL function from your R environment and apply it to the data defined in *featureEngineeringQuery*.

```

featureDataSource = RxSqlServerData(sqlQuery = featureEngineeringQuery,
    colClasses = c(pickup_longitude = "numeric", pickup_latitude = "numeric",
        dropoff_longitude = "numeric", dropoff_latitude = "numeric",
        passenger_count = "numeric", trip_distance = "numeric",
        trip_time_in_secs = "numeric", direct_distance = "numeric"),
    connectionString = connStr)

```

6. Now that the new feature is created, call **rxGetVarsInfo** to create a summary of the data in the feature table.

```
rxGetVarInfo(data = featureDataSource)
```

Results

```

Var 1: tipped, Type: integer
Var 2: fare_amount, Type: numeric
Var 3: passenger_count, Type: numeric
Var 4: trip_time_in_secs, Type: numeric
Var 5: trip_distance, Type: numeric
Var 6: pickup_datetime, Type: character
Var 7: dropoff_datetime, Type: character
Var 8: direct_distance, Type: numeric
Var 9: pickup_latitude, Type: numeric
Var 10: pickup_longitude, Type: numeric
Var 11: dropoff_latitude, Type: numeric
Var 12: dropoff_longitude, Type: numeric

```

NOTE

In some cases, you might get an error like this one: *The EXECUTE permission was denied on the object 'fnCalculateDistance'* If so, make sure that the login you are using has permissions to run scripts and create objects on the database, not just on the instance. Check the schema for the object, fnCalculateDistance. If the object was created by the database owner, and your login belongs to the role db_datareader, you need to give the login explicit permissions to run the script.

Comparing R functions and SQL functions

Remember this piece of code used to time the R code?

```
start.time <- proc.time()  
<your code here>  
used.time <- proc.time() - start.time  
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2), " seconds, Elapsed Time=",  
round(used.time[3],2), " seconds to generate features.", sep=""))
```

You can try using this with the SQL custom function example to see how long the data transformation takes when calling a SQL function. Also, try switching compute contexts with rxSetComputeContext and compare the timings.

Your times might vary significantly, depending on your network speed, and your hardware configuration. In the configurations we tested, the Transact-SQL function approach was faster than using a custom R function. Therefore, we've use the Transact-SQL function for these calculations in subsequent steps.

TIP

Very often, feature engineering using Transact-SQL will be faster than R. For example, T-SQL includes fast windowing and ranking functions that can be applied to common data science calculations such as rolling moving averages and n -tiles. Choose the most efficient method based on your data and task.

Next lesson

[Build an R model and save to SQL](#)

Previous lesson

[View and summarize data using R](#)

Build an R model and save to SQL Server

4/11/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

In this step, you'll learn how to build a machine learning model and save the model in SQL Server.

Create a classification model using rxLogit

The model you build is a binary classifier that predicts whether the taxi driver is likely to get a tip on a particular ride or not. You'll use the data source you created in the previous lesson to train the tip classifier, using logistic regression.

1. Call the `rxLogit` function, included in the **RevoScaleR** package, to create a logistic regression model.

```
system.time(logitObj <- rxLogit(tipped ~ passenger_count + trip_distance + trip_time_in_secs +
direct_distance, data = sql_feature_ds));
```

The call that builds the model is enclosed in the `system.time` function. This lets you get the time required to build the model.

2. After you build the model, you can inspect it using the `summary` function, and view the coefficients.

```
summary(logitObj);
```

Results

*Logistic Regression Results for: tipped ~ passenger_count + trip_distance + trip_time_in_secs + direct_distance**

Data: featureDataSource (RxSqlServerData Data Source)

Dependent variable(s): tipped

Total independent variables: 5

Number of valid observations: 17068

Number of missing observations: 0

*-2*LogLikelihood: 23540.0602 (Residual deviance on 17063 degrees of freedom)*

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) -2.509e-03 3.223e-02 -0.078 0.93793

passenger_count -5.753e-02 1.088e-02 -5.289 1.23e-07 ***

trip_distance -3.896e-02 1.466e-02 -2.658 0.00786 **

trip_time_in_secs 2.115e-04 4.336e-05 4.878 1.07e-06 ***

direct_distance 6.156e-02 2.076e-02 2.966 0.00302 **

*Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.0.1 ' ' 1*

Condition number of final variance-covariance matrix: 48.3933

Number of iterations: 4

Use the logistic regression model for scoring

Now that the model is built, you can use to predict whether the driver is likely to get a tip on a particular drive or not.

1. First, use the [RxSqlServerData](#) function to define a data source object for storing the scoring result

```
scoredOutput <- RxSqlServerData(  
  connectionString = connStr,  
  table = "taxiScoreOutput" )
```

- To make this example simpler, the input to the logistic regression model is the same feature data source (`sql_feature_ds`) that you used to train the model. More typically, you might have some new data to score with, or you might have set aside some data for testing vs. training.
- The prediction results will be saved in the table, `taxiscoreOutput`. Notice that the schema for this table is not defined when you create it using `rxSqlServerData`. The schema is obtained from the `rxPredict` output.
- To create the table that stores the predicted values, the SQL login running the `rxSqlServerData` function must have DDL privileges in the database. If the login cannot create tables, the statement fails.

2. Call the [rxPredict](#) function to generate results.

```
rxPredict(modelObject = logitObj,  
  data = sql_feature_ds,  
  outData = scoredOutput,  
  predVarNames = "Score",  
  type = "response",  
  writeModelVars = TRUE, overwrite = TRUE)
```

If the statement succeeds, it should take some time to run. When complete, you can open SQL Server Management Studio and verify that the table was created and that it contains the Score column and other expected output.

Plot model accuracy

To get an idea of the accuracy of the model, you can use the [rxRoc](#) function to plot the Receiver Operating Curve. Because `rxRoc` is one of the new functions provided by the `RevoScaleR` package that supports remote compute contexts, you have two options:

- You can use the `rxRoc` function to execute the plot in the remote compute context and then return the plot to your local client.
- You can also import the data to your R client computer, and use other R plotting functions to create the performance graph.

In this section, you'll experiment with both techniques.

Execute a plot in the remote (SQL Server) compute context

1. Call the function `rxRoc` and provide the data defined earlier as input.

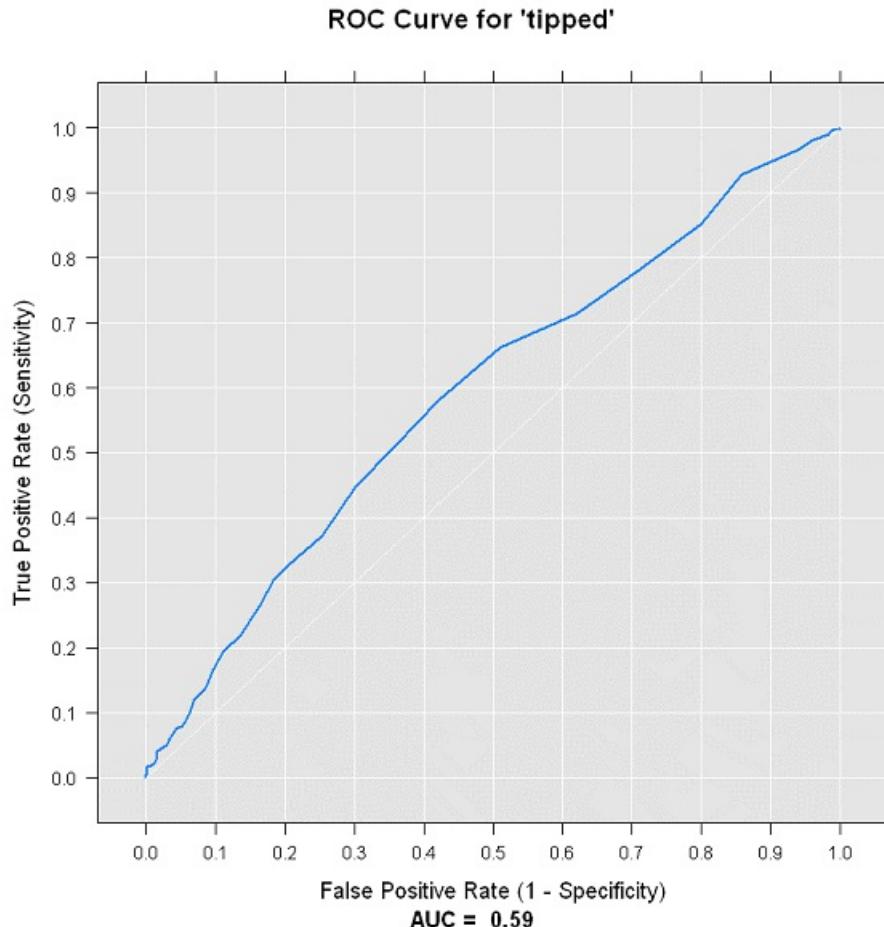
```
scoredOutput = rxImport(scoredOutput);  
rxRoc(actualVarName= "tipped", predVarNames = "Score", scoredOutput);
```

This call returns the values used in computing the ROC chart. The label column is `tipped`, which has the actual results you are trying to predict, while the `Score` column has the prediction.

2. To actually plot the chart, you can save the ROC object and then draw it with the `plot` function. The graph is created on the remote compute context, and returned to your R environment.

```
scoredOutput = rxImport(scoredOutput);
rocObjectOut <- rxRoc(actualVarName= "tipped", predVarNames = "Score", scoredOutput);
plot(rocObjectOut);
```

View the graph by opening the R graphics device, or by clicking the **Plot** window in RStudio.



Create the plots in the local compute context using data from SQL Server

- For the local compute context, the process is much the same. You use the `rxImport` function to bring the specified data into your local R environment.

```
scoredOutput = rxImport(scoredOutput)
```

- Using the data in local memory, you load the **ROCR** package, and use the prediction function from that package to create some new predictions.

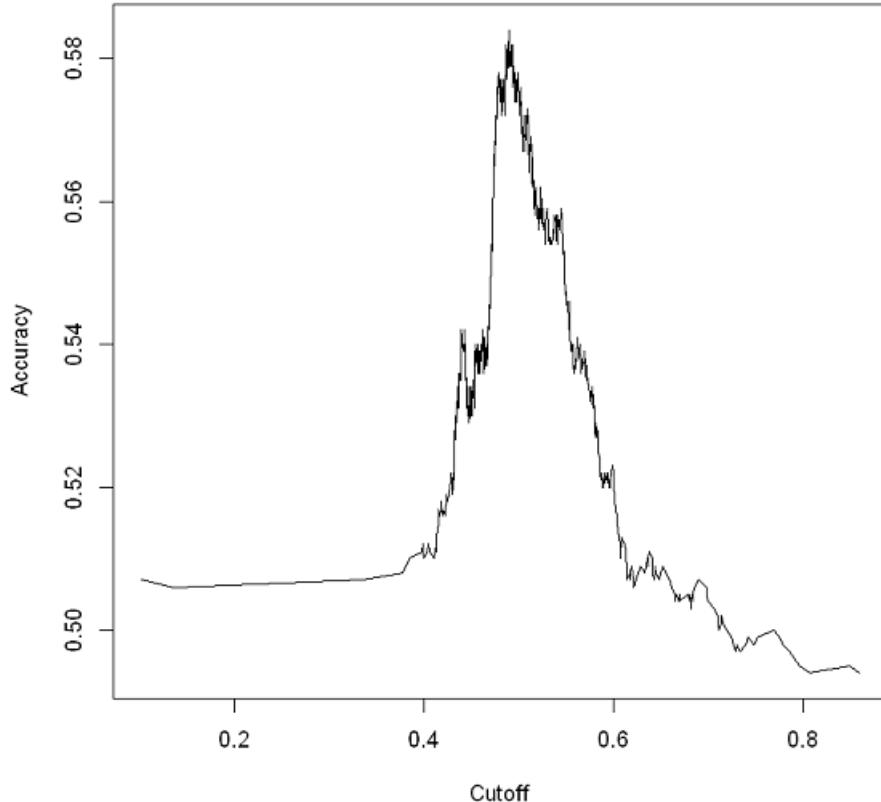
```
library('ROCR');
pred <- prediction(scoredOutput$Score, scoredOutput$tipped);
```

- Generate a local plot, based on the values stored in the output variable `pred`.

```

acc.perf = performance(pred, measure = 'acc');
plot(acc.perf);
ind = which.max( slot(acc.perf, 'y.values')[[1]] );
acc = slot(acc.perf, 'y.values')[[1]][ind];
cutoff = slot(acc.perf, 'x.values')[[1]][ind];

```



NOTE

Your charts might look different from these, depending on how many data points you used.

Deploy the model

After you have built a model and ascertained that it is performing well, you probably want to deploy it to a site where users or people in your organization can make use of the model, or perhaps retrain and recalibrate the model on a regular basis. This process is sometimes called *operationalizing* a model.

Because R Services (In-Database) lets you invoke an R model using a Transact-SQL stored procedure, it is easy to use R in a client application.

However, before you can call the model from an external application, you must save the model to the database used for production. In R Services (In-Database), trained models are stored in binary form, in a single column of type **varbinary(max)**.

Therefore, moving a trained model from R to SQL Server includes these steps:

- Serializing the model into a hexadecimal string
- Transmitting the serialized object to the database
- Saving the model in a varbinary(max) column

In this section, you learn how to persist the model, and how to call it to make predictions.

1. Switch back to your local R environment if you are not already using it, serialize the model, and save it in a variable.

```
rxSetComputeContext("local");
modelbin <- serialize(logitObj, NULL);
modelbinstr=paste(modelbin, collapse="");
```

2. Open an ODBC connection using **RODBC**.

```
library(RODBC);
conn <- odbcDriverConnect(connStr);
```

You can omit the call to RODBC if you already have the package loaded.

3. Call the stored procedure created by the PowerShell script, to store the binary representation of the model in a column in the database.

```
q <- paste("EXEC PersistModel @m='", modelbinstr,"'", sep="");
sqlQuery (conn, q);
```

Saving a model to a table requires only an INSERT statement. However, it's easier when wrapped in a stored procedure, such as *PersistModel*.

NOTE

If you get an error such as "The EXECUTE permission was denied on the object PersistModel", make sure that your login has permission. You can grant explicit permissions on just the stored procedure by running a T-SQL statement like this: `GRANT EXECUTE ON [dbo].[PersistModel] TO <user_name>`

4. After you have created a model and saved it in a database, you can call it directly from Transact-SQL code, using the system stored procedure, [sp_execute_external_script](#).

However, with any model you use often, it's easier to wrap the input query and the call to the model, together with other parameters, in a custom stored procedure.

Here is the complete code of one such stored procedure. We recommend creating stored procedure such as this one to make it easier to manage and update your R models in SQL Server.

```
CREATE PROCEDURE [dbo].[PersistModel]  @m nvarchar(max)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO nyc_taxi_models (model) values (convert(varbinary(max),@m,2))
END
```

NOTE

Use the **SET NOCOUNT ON** clause to prevent extra result sets from interfering with SELECT statements.

In the next and final lesson, you learn how to perform scoring against the saved model using Transact-SQL.

Next lesson

[Deploy the R model and use in SQL](#)

Previous lesson

[Create data features using R and SQL](#)

Deploy the R model and use it in SQL

7/16/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

In this lesson, you use your R models in a production environment, by calling a trained model from a stored procedure. You can then invoke the stored procedure from R or any application programming language that supports Transact-SQL (such as C#, Java, Python, etc.), to use the model to make predictions on new observations.

This sample demonstrates the two most common ways to use a model in scoring:

- **Batch scoring mode** is used when you need to create multiple predictions very fast, by passing a SQL query or table as input. A table of results is returned, which you might insert directly into a table or write to a file.
- **Individual scoring mode** is used when you need to create predictions one at a time. You pass a set of individual values to the stored procedure. The values correspond to features in the model, which the model uses to create a prediction, or generate another result such as a probability value. You can then return that value to the application, or user.

Batch scoring

A stored procedure for batch scoring was created when you initially ran the PowerShell script. This stored procedure, *PredictTipBatchMode*, does the following:

- Gets a set of input data as a SQL query
- Calls the trained logistic regression model that you saved in the previous lesson
- Predicts the probability that the driver gets any non-zero tip

1. Take a minute to look over the script for the stored procedure, *PredictTipBatchMode*. It illustrates several aspects of how a model can be operationalized using R Services (In-Database).

```
CREATE PROCEDURE [dbo].[PredictTipBatchMode]
@input nvarchar(max)
AS
BEGIN
    DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM nyc_taxi_models);
    EXEC sp_execute_external_script @language = N'R',
        @script = N'
            mod <- unserialize(as.raw(model));
            print(summary(mod))
            OutputDataSet<-rxPredict(modelObject = mod,
                data = InputDataSet,
                outData = NULL,
                predVarNames = "Score", type = "response",
                writeModelVars = FALSE, overwrite = TRUE);
            str(OutputDataSet)
            print(OutputDataSet)',
        @input_data_1 = @input,
        @params = N'@model varbinary(max)',
        @model = @lmodel2
        WITH RESULT SETS ((Score float));
END
```

- You use a SELECT statement to call the stored model from a SQL table. The model is retrieved from

the table as **varbinary(max)** data, stored in the SQL variable `@lmodel2`, and passed as the parameter *mod* to the system stored procedure [sp_execute_external_script](#).

- The data used as inputs for scoring is defined as a SQL query and stored as a string in the SQL variable `@input`. As data is retrieved from the database, it is stored in a data frame called `InputDataSet`, which is just the default name for input data to the [sp_execute_external_script](#) procedure; you can define another variable name if needed by using the parameter `@input_data_1_name`.
 - To generate the scores, the stored procedure calls the `rxPredict` function from the **RevoScaleR** library.
 - The return value, *Score*, is the probability, given the model, that driver gets a tip. Optionally, you could easily apply some kind of filter to the returned values to categorize the return values into "tip" and "no tip" groups. For example, a probability of less than 0.5 would mean a tip is unlikely.
2. To call the stored procedure in batch mode, you define the query required as input to the stored procedure. Here is the SQL query; you can run it in SSMS to verify that it works.

```
SELECT TOP 10
    a.passenger_count AS passenger_count,
    a.trip_time_in_secs AS trip_time_in_secs,
    a.trip_distance AS trip_distance,
    a.dropoff_datetime AS dropoff_datetime,
    dbo.fnCalculateDistance( pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS
    direct_distance
    FROM
        (SELECT medallion, hack_license, pickup_datetime, passenger_count,trip_time_in_secs,trip_distance,
        dropoff_datetime, pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
        FROM nyctaxi_sample)a
    LEFT OUTER JOIN
        ( SELECT medallion, hack_license, pickup_datetime
        FROM nyctaxi_sample  tablesample (1 percent) repeatable (98052)  )b
    ON a.medallion=b.medallion
    AND a.hack_license=b.hack_license
    AND a.pickup_datetime=b.pickup_datetime
    WHERE b.medallion is null
```

3. Use this R code to create the input string from the SQL query:

```
input <- "N'SELECT TOP 10 a.passenger_count AS passenger_count, a.trip_time_in_secs AS
    trip_time_in_secs, a.trip_distance AS trip_distance, a.dropoff_datetime AS dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS
    direct_distance FROM (SELECT medallion, hack_license, pickup_datetime,
    passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime, pickup_latitude, pickup_longitude,
    dropoff_latitude, dropoff_longitude FROM nyctaxi_sample)a LEFT OUTER JOIN ( SELECT medallion,
    hack_license, pickup_datetime FROM nyctaxi_sample  tablesample (1 percent) repeatable (98052)  )b ON
    a.medallion=b.medallion AND a.hack_license=b.hack_license AND a.pickup_datetime=b.pickup_datetime WHERE
    b.medallion is null'";
q <- paste("EXEC PredictTipBatchMode @inquery = ", input, sep="");
```

4. To run the stored procedure from R, call the **sqlQuery** method of the **RODBC** package and use the SQL connection `conn` that you defined earlier:

```
sqlQuery (conn, q);
```

If you get an ODBC error, check the query syntax, and whether you have the right number of quotation marks.

If you get a permissions error, make sure the login has the ability to execute the stored procedure.

Single row scoring

When calling the model for prediction on a row-by-row basis, you pass a set of values that represent features for each individual case. The stored procedure then returns a single prediction or probability.

The stored procedure *PredictTipSingleMode* demonstrates this approach. It takes as input multiple parameters representing feature values (for example, passenger count and trip distance), scores these features using the stored R model, and outputs the tip probability.

1. If the stored procedure *PredictTipSingleMode* was not created by the initial PowerShell script, you can run the following Transact-SQL statement to create it now.

```
CREATE PROCEDURE [dbo].[PredictTipSingleMode] @passenger_count int = 0,
@trip_distance float = 0,
@trip_time_in_secs int = 0,
@pickup_latitude float = 0,
@pickup_longitude float = 0,
@dropoff_latitude float = 0,
@dropoff_longitude float = 0
AS
BEGIN
    DECLARE @inquery nvarchar(max) = N'
        SELECT * FROM [dbo].[fnEngineerFeatures](@passenger_count, @trip_distance, @trip_time_in_secs,
@pickup_latitude, @pickup_longitude, @dropoff_latitude, @dropoff_longitude)'
    DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM nyc_taxi_models);

    EXEC sp_execute_external_script @language = N'R',
        @script = N'
            mod <- unserialize(as.raw(model));
            print(summary(mod))
            OutputDataSet<-rxPredict(
                modelObject = mod,
                data = InputDataSet,
                outData = NULL,
                predVarNames = "Score",
                type = "response",
                writeModelVars = FALSE,
                overwrite = TRUE);
            str(OutputDataSet)
            print(OutputDataSet)
        ',
        @input_data_1 = @inquery,
        @params = N'
        -- passthrough columns
        @model varbinary(max) ,
        @passenger_count int ,
        @trip_distance float ,
        @trip_time_in_secs int ,
        @pickup_latitude float ,
        @pickup_longitude float ,
        @dropoff_latitude float ,
        @dropoff_longitude float',
        -- mapped variables
        @model = @lmodel2 ,
        @passenger_count =@passenger_count ,
        @trip_distance=@trip_distance ,
        @trip_time_in_secs=@trip_time_in_secs ,
        @pickup_latitude=@pickup_latitude ,
        @pickup_longitude=@pickup_longitude ,
        @dropoff_latitude=@dropoff_latitude ,
        @dropoff_longitude=@dropoff_longitude
    WITH RESULT SETS ((Score float));
END
```

2. In SQL Server Management Studio, you can use the Transact-SQL **EXEC** procedure (or **EXECUTE**) to call the stored procedure, and pass it the required inputs. For example, try running this statement in Management Studio:

```
EXEC [dbo].[PredictTipSingleMode] 1, 2.5, 631, 40.763958,-73.973373, 40.782139,-73.977303
```

The values passed in here are, respectively, for the variables *passenger_count*, *trip_distance*, *trip_time_in_secs*, *pickup_latitude*, *pickup_longitude*, *dropoff_latitude*, and *dropoff_longitude*.

3. To run this same call from R code, you simply define an R variable that contains the entire stored procedure call, like this one:

```
q2 = "EXEC PredictTipSingleMode 1, 2.5, 631, 40.763958,-73.973373, 40.782139,-73.977303 ";
```

The values passed in here are, respectively, for the variables *passenger_count*, *trip_distance*, *trip_time_in_secs*, *pickup_latitude*, *pickup_longitude*, *dropoff_latitude*, and *dropoff_longitude*.

4. Call `sqlQuery` (from the **RODBC** package) and pass the connection string, together with the string variable containing the stored procedure call.

```
# predict with stored procedure in single mode  
sqlQuery (conn, q2);
```

TIP

R Tools for Visual Studio (RTVS) provides great integration with both SQL Server and R. See this article for more examples of using RODBC with a SQL Server connection: [Working with SQL Server and R](#)

Summary

Now that you have learned how to work with SQL Server data and persist trained R models to SQL Server, it should be relatively easy for you to create new models based on this data set. For example, you might try creating these additional models:

- A regression model that predicts the tip amount
- A multiclass classification model that predicts whether the tip is big, medium, or small

We also recommend that you check out some of these additional samples and resources:

- [Data science scenarios and solution templates](#)
- [In-database advanced analytics](#)
- [Microsoft R - Diving into Data Analysis](#)
- [Additional Resources](#)

Previous lesson

[Build an R model and save it in SQL Server](#)

Next steps

[SQL Server R tutorials](#)

How to create a stored procedure using sqlrutils

Tutorial: Use RevoScaleR R functions with SQL Server data

7/16/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

RevoScaleR is a Microsoft R package providing distributed and parallel processing for data science and machine learning workloads. For R development in SQL Server, RevoScaleR is one of the core built-in packages, with functions for setting a compute context, managing packages, and most importantly: working with data end-to-end, from import to visualization and analysis. Machine Learning algorithms in SQL Server have a dependency on RevoScaleR data sources. Given the importance of RevoScaleR, knowing when and how to call its functions is an essential skill.

In this tutorial, you will learn how to create a remote compute context, move data between local and remote compute contexts, and execute R code on a remote SQL Server. You also learn how to analyze and plot data both locally and on the remote server, and how to create and deploy models.

- Data is initially obtained from CSV files or XDF files. You import the data into SQL Server using the functions in the RevoScaleR package.
- Model training and scoring is performed using the SQL Server compute context.
- Use RevoScaleR functions to create new SQL Server tables to save your scoring results.
- Create plots both on the server and in the local compute context.
- Train a model on data in SQL Server database, running R in the SQL Server instance.
- Extract a subset of data and save it as an XDF file for re-use in analysis on your local workstation.
- Get new data for scoring, by opening an ODBC connection to the SQL Server database. Scoring is done on the local workstation.
- Create a custom R function and run it in the server compute context to perform a simulation.

Target audience

This tutorial is intended for data scientists or for people who are already somewhat familiar with R, and with data science tasks such as summaries and model creation. However, all the code is provided, so even if you are new to R, you can run the code and follow along, assuming you have the required server and client environments.

You should also be comfortable with Transact-SQL syntax and know how to access a SQL Server database using tools such as these:

- SQL Server Management Studio
- Database tools in Visual Studio
- The free [mssql extension for Visual Studio Code](#).

TIP

Save your R workspace between lessons, so that you can easily pick up where you left off.

Prerequisites

- **SQL Server with support for R**

Install [SQL Server 2017 Machine Learning Services](#) with the R feature, or install [SQL Server 2016 R Services \(in-Database\)](#).

Make sure external scripting is enabled, Launchpad service is running, and that you have permissions to access the service.

- **Database permissions**

To run the queries used to train the model, you must have **db_datareader** privileges on the database where the data is stored. To run R, your user must have the permission, EXECUTE ANY EXTERNAL SCRIPT.

- **Data science development computer**

To switch back and forth between local and remote compute contexts, you need two systems. Local is typically a development workstation with sufficient power for data science workloads. Remote in this case is SQL Server 2017 or SQL Server 2016 with the R feature enabled.

Switching compute contexts is predicated on having the same-version RevoScaleR on both local and remote systems. On a local workstation, you can get the RevoScaleR packages and related providers by installing or using any one of the following: [Data Science VM on Azure](#), [Microsoft R Client \(free\)](#), or [Microsoft Machine Learning Server \(Standalone\)](#). For the standalone server option, install the free developer edition, using either Linux or Windows installers. You can also use SQL Server Setup to install a standalone server.

- **Additional R Packages**

In this tutorial, you install the following packages: **dplyr**, **ggplot2**, **ggthemes**, **reshape2**, and **e1071**. Instructions are provided as part of the tutorial.

All packages must be installed in two places: on the workstation used for R solution development, and on the SQL Server computer where R scripts are executed. If you do not have permission to install packages on the server computer, ask an administrator.

Do not install the packages to a user library. The packages must be installed in the R package library that is used by the SQL Server instance.

R development tools

R developers typically use IDEs for writing and debugging R code. Here are some suggestions:

- **R Tools for Visual Studio** (RTVS) is a free plug-in that provides Intellisense, debugging, and support for Microsoft R. You can use it with both R Server and SQL Server Machine Learning Services. To download, see [R Tools for Visual Studio](#).
- **RStudio** is one of the more popular environments for R development. For more information, see <https://www.rstudio.com/products/RStudio/>.
- Basic R tools (R.exe, RTerm.exe, RScripts.exe) are also installed by default when you install R in SQL Server or R Client. If you do not wish to install an IDE, you can use built-in R tools to execute the code in this tutorial.

Recall that RevoScaleR is required on both local and remote computers. You cannot complete this tutorial using a generic installation of RStudio or other environment that's missing the Microsoft R libraries. For more information, see [Set Up a Data Science Client](#).

Next steps

Lesson 1: Create database and permissions

Lesson 1: Create a database and permissions

7/16/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the [RevoScaleR tutorial](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this lesson, you set up the environment and add the data you need for training your models and run some quick summaries of the data. As part of the process, you must complete these tasks:

- Create a new database to store the data for training and scoring two R models.
- Create an account (either a Windows user or SQL login) to use when communicating between your workstation and the SQL Server computer.
- Create data sources in R for working with SQL Server data and database objects.
- Use the R data source to load data into SQL Server.
- Use R to get a list of variables and modify the metadata of the SQL Server table.
- Create a compute context to enable remote execution of R code.
- (Optional) Enable tracing on the remote compute context.

Create the database and user

For this walkthrough, create a new database in SQL Server 2017, and add a SQL login with permissions to write and read data, and to run R scripts.

NOTE

If you are only reading data, the account that runs the R scripts requires SELECT permissions (**db_datareader** role) on the specified database. However, in this tutorial, you must have DDL admin privileges to prepare the database, and to create tables for saving the scoring results.

Additionally, if you are not the database owner, you need the permission, EXECUTE ANY EXTERNAL SCRIPT, in order to execute R scripts.

1. In SQL Server Management Studio, select the instance where R Services (In-Database) is enabled, right-click **Databases**, and select **New database**.

2. Type a name for the new database. You can use any name you want; just remember to edit all the Transact-SQL scripts and R scripts in this walkthrough accordingly.

TIP

To view the updated database name, right-click **Databases** and select **Refresh**.

3. Click **New Query**, and change the database context to the master database.

4. In the new **Query** window, run the following commands to create the user accounts and assign them to the database used for this tutorial. Be sure to change the database name if needed.

Windows user

```
-- Create server user based on Windows account
USE master
GO
CREATE LOGIN [<DOMAIN>\<user_name>] FROM WINDOWS WITH DEFAULT_DATABASE=[DeepDive]

--Add the new user to tutorial database
USE [DeepDive]
GO
CREATE USER [<user_name>] FOR LOGIN [<DOMAIN>\<user_name>] WITH DEFAULT_SCHEMA=[db_datareader]
```

SQL login

```
-- Create new SQL login
USE master
GO
CREATE LOGIN DDUser01 WITH PASSWORD='<type password here>', CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF;

-- Add the new SQL login to tutorial database
USE [DeepDive]
GO
CREATE USER [DDUser01] FOR LOGIN [DDUser01] WITH DEFAULT_SCHEMA=[db_datareader]
```

1. To verify that the user has been created, select the new database, expand **Security**, and expand **Users**.

Troubleshooting

This section lists some common issues that you might run across in the course of setting up the database.

- **How can I verify database connectivity and check SQL queries?**

Before you run R code using the server, you might want to check that the database can be reached from your R development environment. Both [Server Explorer in Visual Studio](#) and [SQL Server Management Studio](#) are free tools with powerful database connectivity and management features.

If you don't want to install additional database management tools, you can create a test connection to the SQL Server instance by using the [ODBC Data Source Administrator](#) in Control Panel. If the database is configured correctly and you enter the correct user name and password, you should be able to see the database you just created and select it as your default database.

If you cannot connect to the database, verify that remote connections are enabled for the server, and that the Named Pipes protocol has been enabled. Additional troubleshooting tips are provided in this article: [Troubleshoot Connecting to the SQL Server Database Engine](#).

- **My table name has datareader prefixed to it - why?**

When you specify the default schema for this user as **db_datareader**, all tables and other new objects created by this user are prefixed with the *schema* name. A schema is like a folder that you can add to a database to organize objects. The schema also defines a user's privileges within the database.

When the schema is associated with one particular user name, the user is the *schema owner*. When you create an object, you always create it in your own schema, unless you specifically ask it to be created in another schema.

For example, if you create a table with the name `* TestData`

`, and your default schema is **db__datareader**, the table is created with the name <database_name>.db_datareader.TestData`.`

For this reason, a database can contain multiple tables with the same names, as long as the tables belong to different schemas.

If you are looking for a table and do not specify a schema, the database server looks for a schema that you own. Therefore, there is no need to specify the schema name when accessing tables in a schema associated with your login.

- **I don't have DDL privileges. Can I still run the tutorial??**

Yes; however, you should ask someone to pre-load the data into the SQL Server tables, and skip past the sections that call for creating new tables. The functions that require DDL privileges are called out in the tutorial wherever possible.

Also, ask your administrator to grant you the permission, EXECUTE ANY EXTERNAL SCRIPT. It is needed for R script execution, whether remote or by using `sp_execute_external_script`.

Next step

[Create SQL Server data objects using RxSqlServerData](#)

Overview

[Data Science Deep Dive: Using the RevoScaleR Packages](#)

Create SQL Server data objects using RxSqlServerData (SQL and R deep dive)

4/11/2018 • 6 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

By now you have created the SQL Server database and have the necessary permissions to work with the data. In this step, you create some objects in R that let you work with the data.

Create the SQL Server data objects

In this step, you use functions from the **RevoScaleR** package to create and populate two tables. One table is used for training the models, and the other table is used for scoring. Both tables contain simulated credit card fraud data.

To create tables on the remote SQL Server computer, call the **RxSqlServerData** function.

TIP

If you're using R Tools for Visual Studio, select **R Tools** from the toolbar and click **Windows** to see options for debugging and viewing R variables.

Create the training data table

1. Save the database connection string in an R variable. Here are two examples of valid ODBC connection strings for SQL Server: one using a SQL login, and one for Windows integrated authentication.

SQL login

```
sqlConnString <- "Driver=SQL Server;Server=instance_name; Database=DeepDive;Uid=user_name;Pwd=password"
```

Windows authentication

```
sqlConnString <- "Driver=SQL Server;Server=instance_name;Database=DeepDive;Trusted_Connection=True"
```

Be sure to modify the instance name, database name, user name, and password as appropriate.

2. Specify the name of the table you want to create, and save it in an R variable.

```
sqlFraudTable <- "ccFraudSmall"
```

Because the instance and database name are already specified as part of the connection string, when you combine the two variables, the *fully qualified* name of the new table becomes *instance.database.schema.ccFraudSmall*.

3. Before instantiating the data source object, add a line specifying an additional parameter, *rowsPerRead*. The *rowsPerRead* parameter controls how many rows of data are read in each batch.

```
sqlRowsPerRead = 5000
```

Although this parameter is optional, it is important for handling memory usage and efficient computations. Most of the enhanced analytical functions in R Services (In-Database) process data in chunks and store intermediate results, returning the final computations after all of the data has been read.

If the value of this parameter is too large, data access might be slow because you don't have enough memory to efficiently process such a large chunk of data. On some systems, if the value of *rowsPerRead* is too small, performance might be slower. Therefore, we recommend that you experiment with this setting on your system when you are working with a large data set.

For this walkthrough, use the default batch process size defined by the SQL Server instance to control the number of rows in each chunk. Save that value in the variable `sqlRowsPerRead`.

- Finally, define a variable for the new data source object, and pass the arguments previously defined to the `RxSqlServerData` constructor. Note that this only creates the data source object and does not populate it.

```
sqlFraudDS <- RxSqlServerData(connectionString = sqlConnString,  
    table = sqlFraudTable,  
    rowsPerRead = sqlRowsPerRead)
```

To create the scoring data table

Using the same steps, create the table that holds the scoring data using the same process.

- Create a new R variable, `sqlScoreTable`, to store the name of the table used for scoring.

```
sqlScoreTable <- "ccFraudScoreSmall"
```

- Provide that variable as an argument to the `RxSqlServerData` function to define a second data source object, `sqlScoreDS`.

```
sqlScoreDS <- RxSqlServerData(connectionString = sqlConnString,  
    table = sqlScoreTable, rowsPerRead = sqlRowsPerRead)
```

Because you've already defined the connection string and other parameters as variables in the R workspace, it is easy to create new data sources for different tables, views, or queries.

NOTE

The function uses different arguments for defining a data source based on an entire table than for a data source based on a query. This is because the SQL Server database engine must prepare the queries differently. Later in this tutorial, you learn how to create a data source object based on a SQL query.

Load data into SQL tables using R

Now that you have created the SQL Server tables, you can load data into them using the appropriate **Rx** function.

The **RevoScaleR** package contains functions that support many different data sources: For text data, use `RxTextData` to generate the data source object. There are additional functions for creating data source objects from Hadoop data, ODBC data, and so forth.

NOTE

For this section, you must have **Execute DDL** permissions on the database.

Load data into the training table

1. Create an R variable, `ccFraudCsv`, and assign to the variable the file path for the CSV file containing the sample data.

```
ccFraudCsv <- file.path(rxGetOption("sampleDataDir"), "ccFraudSmall.csv")
```

Notice the call to **rxGetOption**, which is the GET method associated with **rxOptions** in **RevoScaleR**. Use this utility to set and list options related to local and remote compute contexts, such as the default shared directory, or the number of processors (cores) to use in computations.

This particular call gets the samples from the correct library, regardless of where you are running your code. For example, try running the function on SQL Server, and on your development computer, and see how the paths differ.

2. Define a variable to store the new data, and use the **RxTextData** function to specify the text data source.

```
inTextData <- RxTextData(file = ccFraudCsv, colClasses = c(  
    "custID" = "integer", "gender" = "integer", "state" = "integer",  
    "cardholder" = "integer", "balance" = "integer",  
    "numTrans" = "integer",  
    "numIntlTrans" = "integer", "creditLine" = "integer",  
    "fraudRisk" = "integer"))
```

The argument `colClasses` is important. You use it to indicate the data type to assign to each column of data loaded from the text file. In this example, all columns are handled as text, except for the named columns, which are handled as integers.

3. At this point, you might want to pause a moment, and view your database in SQL Server Management Studio. Refresh the list of tables in the database.

You can see that, although the R data objects have been created in your local workspace, the tables have not been created in the SQL Server database. Also, no data has been loaded from the text file into the R variable.

4. Now, call the function **rxDataStep** to insert the data into the SQL Server table.

```
rxDataStep(inData = inTextData, outFile = sqlFraudDS, overwrite = TRUE)
```

Assuming no problems with your connection string, after a brief pause, you should see results like these:

Total Rows written: 10000, Total time: 0.466

Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.577 seconds

5. Using SQL Server Management Studio, refresh the list of tables. To verify that each variable has the correct data types and was imported successfully, you can also right-click the table in SQL Server Management Studio and select **Select Top 1000 Rows**.

Load data into the scoring table

1. Repeat the steps to load the data set used for scoring into the database.

Start by providing the path to the source file.

```
ccScoreCsv <- file.path(rxGetOption("sampleDataDir"), "ccFraudScoreSmall.csv")
```

2. Use the **RxTextData** function to get the data and save it in the variable, *inTextData*.

```
inTextData <- RxTextData(file = ccScoreCsv, colClasses = c(
  "custID" = "integer", "gender" = "integer", "state" = "integer",
  "cardholder" = "integer", "balance" = "integer",
  "numTrans" = "integer",
  "numIntlTrans" = "integer", "creditLine" = "integer"))
```

3. Call the **rxDataStep** function to overwrite the current table with the new schema and data.

```
rxDataStep(inData = inTextData, sqlScoreDS, overwrite = TRUE)
```

- The *inData* argument defines the data source to use.
- The *outFile* argument specifies the table in SQL Server where you want to save the data.
- If the table already exists and you don't use the *overwrite* option, results are inserted without truncation.

Again, if the connection was successful, you should see a message indicating completion and the time required to write the data into the table:

Total Rows written: 10000, Total time: 0.384

Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.456 seconds

More about rxDataStep

[rxDataStep](#) is a powerful function that can perform multiple transformations on an R data frame. You can also use rxDataStep to convert data into the representation required by the destination: in this case, SQL Server.

Optionally, you can specify transformations on the data, by using R functions in the arguments to **rxDataStep**. Examples of these operations are provided later in this tutorial.

Next step

[Query and modify the SQL Server data](#)

Previous step

[Work with SQL Server data using R](#)

Query and modify the SQL Server data (SQL and R deep dive)

4/11/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

Now that you've loaded the data into SQL Server, you can use the data sources you created as arguments to R functions in R Services (In-Database), to get basic information about the variables, and generate summaries and histograms.

In this step, you re-use the data sources to do some quick analysis and then enhance the data.

Query the data

First, get a list of the columns and their data types.

1. Use the function [rxGetVarInfo](#) and specify the data source you want to analyze.

Depending on your version of RevoScaleR, you can also use [rxGetVarNames](#).

```
rxGetVarInfo(data = sqlFraudDS)
```

Results

Var 1: custID, Type: integer
Var 2: gender, Type: integer
Var 3: state, Type: integer
Var 4: cardholder, Type: integer
Var 5: balance, Type: integer
Var 6: numTrans, Type: integer
Var 7: numIntlTrans, Type: integer
Var 8: creditLine, Type: integer
Var 9: fraudRisk, Type: integer

Modify metadata

All the variables are stored as integers, but some variables represent categorical data, called *factor variables* in R. For example, the column *state* contains numbers used as identifiers for the 50 states plus the District of Columbia. To make it easier to understand the data, you replace the numbers with a list of state abbreviations.

In this step, you create a string vector containing the abbreviations, and then map these categorical values to the original integer identifiers. Then you use the new variable in the *collInfo* argument, to specify that this column be handled as a factor. Whenever you analyze the data or move it, the abbreviations are used and the column is handled as a factor.

Mapping the column to abbreviations before using it as a factor actually improves performance as well. For more information, see [R and data optimization](#).

1. Begin by creating an R variable, *stateAbb*, and defining the vector of strings to add to it, as follows:

```
stateAbb <- c("AK", "AL", "AR", "AZ", "CA", "CO", "CT", "DC",
"DE", "FL", "GA", "HI", "IA", "ID", "IL", "IN", "KS", "KY", "LA",
"MA", "MD", "ME", "MI", "MN", "MO", "MS", "MT", "NB", "NC", "ND",
"NH", "NJ", "NM", "NV", "NY", "OH", "OK", "OR", "PA", "RI", "SC",
"SD", "TN", "TX", "UT", "VA", "VT", "WA", "WI", "WV", "WY")
```

2. Next, create a column information object, named *ccColInfo*, that specifies the mapping of the existing integer values to the categorical levels (the abbreviations for states).

This statement also creates factor variables for gender and cardholder.

```
ccColInfo <- list(
  gender = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Male", "Female")
  ),
  cardholder = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Principal", "Secondary")
  ),
  state = list(
    type = "factor",
    levels = as.character(1:51),
    newLevels = stateAbb
  ),
  balance = list(type = "numeric")
)
```

3. To create the SQL Server data source that uses the updated data, call the **RxSqlServerData** function as before but add the *colInfo* argument.

```
sqlFraudDS <- RxSqlServerData(connectionString = sqlConnString,
  table = sqlFraudTable, colInfo = ccColInfo,
  rowsPerRead = sqlRowsPerRead)
```

- For the *table* parameter, pass in the variable *sqlFraudTable*, which contains the data source you created earlier.
- For the *colInfo* parameter, pass in the *ccColInfo* variable, which contains the column data types and factor levels.

4. You can now use the function **rxGetVarInfo** to view the variables in the new data source.

```
rxGetVarInfo(data = sqlFraudDS)
```

Results

Var 1: custID, Type: integer

Var 2: gender 2 factor levels: Male Female

Var 3: state 51 factor levels: AK AL AR AZ CA ... VT WA WI WV WY

Var 4: cardholder 2 factor levels: Principal Secondary

Var 5: balance, Type: integer

Var 6: numTrans, Type: integer

Var 7: numIntlTrans, Type: integer

Var 8: creditLine, Type: integer

Var 9: fraudRisk, Type: integer

Now the three variables you specified (*gender*, *state*, and *cardholder*) are treated as factors.

Next step

[Define and use compute contexts](#)

Previous step

[Create SQL Server data objects using RxSqlServerData](#)

Define and use compute contexts (SQL and R deep dive)

4/11/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

This lesson introduces the [RxInSqlServer](#) function, which lets you define a compute context for SQL Server and then execute complex calculations on the server, rather than your local computer.

RevoScaleR supports multiple compute contexts, so that you can run R code in Hadoop, Spark, or in-database. For SQL Server, you define the server, and the function handles the tasks of creating the database connection and passing objects between the local computer and the remote execution context.

The function that creates the SQL Server compute context uses the following information:

- Connection string for the SQL Server instance
- Specification of how output should be handled
- Optional arguments that enable tracing or specify the trace level
- Optional specification of a shared data directory

Create and set a compute context

1. Specify the connection string for the instance where computations are performed. You can re-use the connection string that you created earlier. You can create a different connection string if you want to move the computations to a different server, or use a different login to perform some tasks.

Using a SQL login

```
sqlConnString <- "Driver=SQL Server;Server=<SQL Server instance name>; Database=<database name>;Uid=<SQL user name>;Pwd=<password>"
```

Using Windows authentication

```
sqlConnString <- "Driver=SQL Server;Server=instance_name;Database=DeepDive;Trusted_Connection=True"
```

2. Specify how you want the output handled. In the following code, you are indicating that the R session on the workstation should always wait for R job results, but not return console output from remote computations.

```
sqlWait <- TRUE  
sqlConsoleOutput <- FALSE
```

The *wait* argument to [RxInSqlServer](#) supports these options:

- **TRUE.** The job is configured as blocking and does not return until it has completed or has failed. For more information, see [Distributed and parallel computing in Machine Learning Server](#).
- **FALSE.** Jobs are configured as non-blocking and return immediately, allowing you to continue

running other R code. However, even in non-blocking mode, the client connection with SQL Server must be maintained while the job is running.

3. Optionally, you can specify the location of a local directory for shared use by the local R session and by the remote SQL Server computer and its accounts.

```
sqlShareDir <- paste("c:\\AllShare\\\", Sys.getenv("USERNAME"), sep="")
```

4. If you want to manually create a specific directory for sharing, you can add a line like the following:

```
dir.create(sqlShareDir, recursive = TRUE)
```

To determine which folder is currently being used for sharing, run `rxGetComputeContext()`, which returns details about the current compute context. For more information, see the [ScaleR reference](#).

5. Having prepared all the variables, provide them as arguments to the **RxInSqlServer** constructor, to create the *compute context object*.

```
sqlCompute <- RxInSqlServer(  
  connectionString = sqlConnString,  
  wait = sqlWait,  
  consoleOutput = sqlConsoleOutput)
```

The syntax for **RxInSqlServer** looks almost identical to that of the **RxSqlServerData** function that you used earlier to define the data source. However, there are some important differences.

- The data source object, defined by using the function **RxSqlServerData**, specifies where the data is stored.
- In contrast, the compute context, defined by using the function **RxInSqlServer** indicates where aggregations and other computations are to take place.

Defining a compute context does not affect any other generic R computations that you might perform on your workstation, and does not change the source of the data. For example, you could define a local text file as the data source but change the compute context to SQL Server and do all your reading and summaries on the data on the SQL Server computer.

Enable tracing on the compute context

Sometimes operations work on your local context, but have issues when running in a remote compute context. If you want to analyze issues or monitor performance, you can enable tracing in the compute context, to support run-time troubleshooting.

1. Create a new compute context that uses the same connection string, but add the arguments *traceEnabled* and *traceLevel* to the **RxInSqlServer** constructor.

```
sqlComputeTrace <- RxInSqlServer(  
  connectionString = sqlConnString,  
  #shareDir = sqlShareDir,  
  wait = sqlWait,  
  consoleOutput = sqlConsoleOutput,  
  traceEnabled = TRUE,  
  traceLevel = 7)
```

In this example, the *traceLevel* property is set to 7, meaning "show all tracing information."

2. To change the compute context, use the `rxSetComputeContext` function and specify the context by name.

```
rxSetComputeContext( sqlComputeTrace)
```

NOTE

For this tutorial, use the compute context that does not have tracing enabled.

However, if you decide to use tracing, be aware that your experience may be affected by network connectivity. Also be aware that because performance for the tracing-enabled option has not been tested for all operations.

In the next step you learn how to use compute contexts, to run R code on the server or locally.

Next step

[Create and run R scripts](#)

Previous step

[Query and modify the SQL Server data](#)

Create and run R scripts (SQL and R deep dive)

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

Now that you've set up your data sources and established one or several compute contexts, you're ready to run some high-powered R scripts using SQL Server. In this lesson, you use the server compute context to do some common machine learning tasks:

- Visualize data and generate some summary statistics
- Create a linear regression model
- Create a logistic regression model
- Score new data and create a histogram of the scores

Change compute context to the server

Before you run any R code, you need to specify the *current* or *active* compute context.

1. To activate a compute context that you've already defined using R, use the **rxSetComputeContext** function as shown here:

```
rxSetComputeContext(sqlCompute)
```

As soon as you run this statement, all subsequent computations take place on the SQL Server computer specified in the *sqlCompute* parameter.

2. If you decide that you'd rather run the R code on your workstation, you can switch the compute context back to the local computer by using the **local** keyword.

```
rxSetComputeContext ("local")
```

For a list of other keywords supported by this function, type `help("rxSetComputeContext")` from an R command line.

3. After you've specified a compute context, it remains active until you change it. However, any R scripts that *cannot* be run in a remote server context will be run locally.

Compute some summary statistics

To see how the compute context works, try generating some summary statistics using the `sqlFrauds` data source. Remember, the data source object just defines the data you use; it doesn't change the compute context.

- To perform the summary locally, use **rxSetComputeContext** and specify the *local* keyword.
 - To create the same calculations on the SQL Server computer, switch to the SQL compute context you defined earlier.
1. Call the **rxSummary** function and pass required arguments, such as the formula and the data source, and assign the results to the variable `sumOut`.

```
sumOut <- rxSummary(formula = ~gender + balance + numTrans + numIntlTrans + creditLine, data =  
sqlFraudDS)
```

The R language provides many summary functions, but **rxSummary** supports execution on various remote compute contexts, including SQL Server. For information about similar functions, see [Data summaries using RevoScaleR](#).

- When processing is done, you can print the contents of the `sumOut` variable to the console.

```
sumOut
```

NOTE

Don't try to print the results before they have returned from the SQL Server computer, or you might get an error.

Results

Summary Statistics Results for: ~gender + balance + numTrans +

numIntlTrans + creditLine

Data: sqlFraudDS (RxSqlServerData Data Source)

Number of valid observations: 10000

Name Mean StdDev Min Max ValidObs MissingObs

balance 4075.0318 3926.558714 0 25626 100000

numTrans 29.1061 26.619923 0 100 10000 0 100000

numIntlTrans 4.0868 8.726757 0 60 10000 0 100000

creditLine 9.1856 9.870364 1 75 10000 0 100000

Category Counts for gender

Number of categories: 2

Number of valid observations: 10000

Number of missing observations: 0

gender Counts

Male 6154

Female 3846

Add maximum and minimum values

Based on the computed summary statistics, you've discovered some useful information about the data that you want to put into the data source for use in further computations. For example, the minimum and maximum values can be used to compute histograms. For this reason, let's add the high and low values to the **RxSqlServerData** data source.

Fortunately R Services (In-Database) includes optimized functions that can efficiently convert integer data to categorical factor data.

1. Start by setting up some temporary variables.

```
sumDF <- sumOut$sDataFrame  
var <- sumDF$name
```

2. Use the variable `ccColInfo` that you created earlier to define the columns in the data source.

Also, add some new computed columns (`numTrans`, `numIntlTrans`, and `creditLine`) to the column collection.

```
ccColInfo <- list(  
  gender = list(type = "factor",  
    levels = c("1", "2"),  
    newLevels = c("Male", "Female")),  
  cardholder = list(type = "factor",  
    levels = c("1", "2"),  
    newLevels = c("Principal", "Secondary")),  
  state = list(type = "factor",  
    levels = as.character(1:51),  
    newLevels = stateAbb),  
  balance = list(type = "numeric"),  
  numTrans = list(type = "factor",  
    levels = as.character(sumDF[var == "numTrans", "Min"] : sumDF[var == "numTrans", "Max"])),  
  numIntlTrans = list(type = "factor",  
    levels = as.character(sumDF[var == "numIntlTrans", "Min"] : sumDF[var == "numIntlTrans", "Max"])),  
  creditLine = list(type = "numeric")  
)
```

3. Having updated the column collection, apply the following statement to create an updated version of the SQL Server data source that you defined earlier.

```
sqlFraudDS <- RxSqlServerData(  
  connectionString = sqlConnString,  
  table = sqlFraudTable,  
  colInfo = ccColInfo,  
  rowsPerRead = sqlRowsPerRead)
```

The `sqlFraudDS` data source now includes the new columns added using `ccColInfo`.

At this point, the modifications affect only the data source object in R; no new data has been written to the database table yet. However, you can use the data captured in the `sumOut` variable to create visualizations and summaries. In the next step you learn how to do this while switching compute contexts.

TIP

If you forget which compute context you're using, run `rxGetComputeContext()`. A return value of "RxLocalSeq Compute Context" indicates that you are running in the local compute context.

Next step

[Visualize SQL Server data using R](#)

Previous step

[Define and use compute contexts](#)

Visualize SQL Server data using R (SQL and R deep dive)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

The enhanced packages in R Services (In-Database) include multiple functions that have been optimized for scalability and parallel processing. Typically these functions are prefixed with **rx** or **Rx**.

For this walkthrough, you use the **rxHistogram** function to view the distribution of values in the *creditLine* column by gender.

Visualize data using rxHistogram

1. Use the following R code to call the **rxHistogram** function and pass a formula and data source. You can run this locally at first, to see the expected results, and how long it takes.

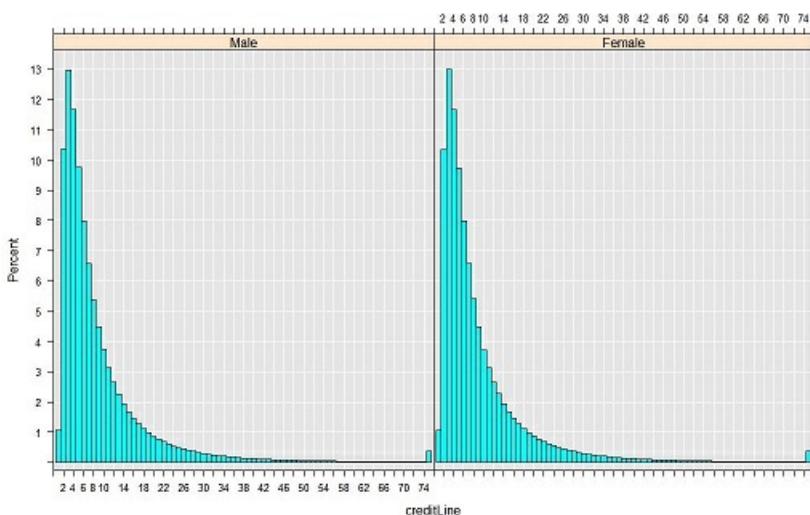
```
rxHistogram(~creditLine|gender, data = sqlFraudDS, histType = "Percent")
```

Internally, **rxHistogram** calls the **rxCube** function, which is included in the **RevoScaleR** package. **rxCube** outputs a single list (or data frame) containing one column for each variable specified in the formula, plus a counts column.

2. Now, set the compute context to the remote SQL Server computer and run **rxHistogram** again.

```
rxSetComputeContext(sqlCompute)
```

3. The results are exactly the same, since you're using the same data source; however, the computations are performed on the SQL Server computer. The results are then returned to your local workstation for plotting.



1. You can also call the **rxCube** function and pass the results to an R plotting function. For example, the following example uses **rxCube** to compute the mean of *fraudRisk* for every combination of *numTrans* and

`numIntlTrans`:

```
cube1 <- rxCube(fraudRisk~F(numTrans):F(numIntlTrans), data = sqlFraudDS)
```

To specify the groups used to compute group means, use the `F()` notation. In this example,

`F(numTrans):F(numIntlTrans)` indicates that the integers in the variables `_numTrans` and `numIntlTrans` should be treated as categorical variables, with a level for each integer value.

Because the low and high levels were already added to the data source `sqlFraudDS` (using the `colInfo` parameter), the levels are automatically used in the histogram.

2. The default return value of **rxCube** is an *rxCube object*, which represents a cross-tabulation. However, you can use the **rxResultsDF** function to convert the results into a data frame that can easily be used in one of R's standard plotting functions.

```
cubePlot <- rxResultsDF(cube1)
```

The **rxCube** function includes an optional argument, `returnDataFrame = TRUE`, that you could use to convert the results to a data frame directly. For example:

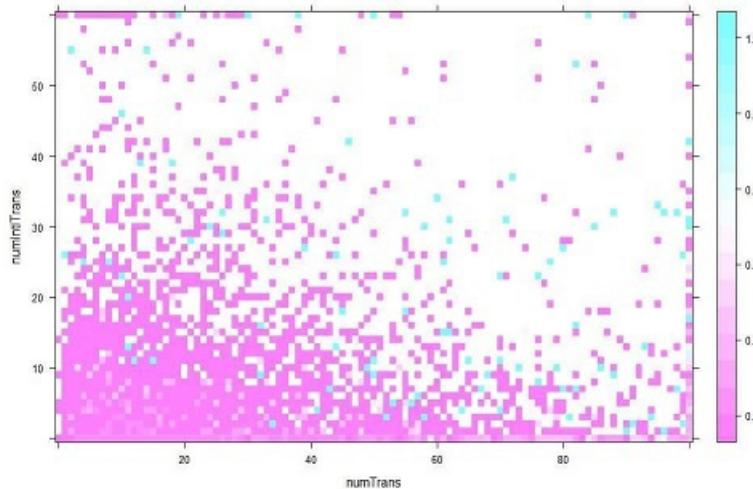
```
print(rxCube(fraudRisk~F(numTrans):F(numIntlTrans), data = sqlFraudDS, returnDataFrame = TRUE))
```

However, the output of **rxResultsDF** is much cleaner and preserves the names of the source columns.

3. Finally, run the following code to create a heat map using the `levelplot` function from the **lattice** package, which is included with all R distributions.

```
levelplot(fraudRisk~numTrans*numIntlTrans, data = cubePlot)
```

Results



From even this quick analysis, you can see that the risk of fraud increases with both the number of transactions and the number of international transactions.

For more information about the **rxCube** function and crosstabs in general, see [Data summaries using RevoScaleR](#).

Next step

[Create R models using SQL Server data](#)

Previous step

[Create and run R scripts](#)

Create R models (SQL and R deep dive)

4/30/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

Now that you have enriched the training data, it's time to analyze the data using linear regression. Linear models are an important tool in the world of predictive analytics, and the **RevoScaleR** package in R Services (In-Database) includes a high-performance, scalable algorithm.

Create a linear regression model

In this step, you create a simple linear model that estimates the credit card balance for the customers, using as independent variables the values in the *gender* and *creditLine* columns.

To do this, use the **rxLinMod** function, which supports remote compute contexts.

1. Create an R variable to store the completed model, and call **rxLinMod**, passing an appropriate formula.

```
linModObj <- rxLinMod(balance ~ gender + creditLine, data = sqlFraudDS)
```

2. To view a summary of the results, call the standard R **summary** function on the model object.

```
summary(linModObj)
```

You might think it peculiar that a plain R function like **summary** would work here, since in the previous step, you set the compute context to the server. However, even when the **rxLinMod** function uses the remote compute context to create the model, it also returns an object that contains the model to your local workstation, and stores it in the shared directory.

Therefore, you can run standard R commands against the model just as if it had been created using the "local" context.

Results

```

Linear Regression Results for: balance ~ gender + creditLineData: sqlFraudDS (RxSqlServerData Data Source)
Dependent variable(s): balance
Total independent variables: 4 (Including number dropped: 1)
Number of valid observations: 10000
Number of missing observations: 0
Coefficients: (1 not defined because of singularities)

Estimate Std. Error t value Pr(>|t|) (Intercept)
3253.575 71.194 45.700 2.22e-16
gender=Male -88.813 78.360 -1.133 0.257
gender=Female Dropped Dropped Dropped Dropped
creditLine 95.379 3.862 24.694 2.22e-16
Signif. codes: 0 0.001 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3812 on 9997 degrees of freedom
Multiple R-squared: 0.05765
Adjusted R-squared: 0.05746
F-statistic: 305.8 on 2 and 9997 DF, p-value: < 2.2e-16
Condition number: 1.0184

```

Create a logistic regression model

Next, you create a logistic regression model that indicates whether a particular customer is a fraud risk. You'll use the **RevoScaleR rxLogit** function, which supports fitting of logistic regression models in remote compute contexts.

1. Keep the compute context as is. You'll also continue to use the same data source as well.
2. Call the **rxLogit** function and pass the formula needed to define the model.

```

logitObj <- rxLogit(fraudRisk ~ state + gender + cardholder + balance + numTrans + numIntlTrans +
creditLine, data = sqlFraudDS, dropFirst = TRUE)

```

Because it is a large model, containing 60 independent variables, including three dummy variables that are dropped, you might have to wait some time for the compute context to return the object.

The reason the model is so large is that, in R and in the **RevoScaleR** package, every level of a categorical factor variable is automatically treated as a separate dummy variable.

3. To view a summary of the returned model, call the R `summary` function.

```

summary(logitObj)

```

Partial results

```

Logistic Regression Results for: fraudRisk ~ state + gender + cardholder + balance + numTrans + numIntlTrans +
creditLine
Data: sqlFraudDS (RxSqlServerData Data Source)
Dependent variable(s): fraudRisk
Total independent variables: 60 (Including number dropped: 3)
Number of valid observations: 10000 -2

LogLikelihood: 2032.8699 (Residual deviance on 9943 degrees of freedom)

Coefficients:
Estimate Std. Error z value Pr(>|z|)      (Intercept)
-8.627e+00  1.319e+00 -6.538 6.22e-11
state=AK          Dropped    Dropped Dropped Dropped
state=AL          -1.043e+00  1.383e+00 -0.754   0.4511

(other states omitted)

gender=Male        Dropped    Dropped Dropped Dropped
gender=Female      7.226e-01  1.217e-01  5.936 2.92e-09
cardholder=Principal Dropped    Dropped Dropped Dropped
cardholder=Secondary 5.635e-01  3.403e-01  1.656  0.0977
balance            3.962e-04  1.564e-05  25.335 2.22e-16
numTrans           4.950e-02  2.202e-03  22.477 2.22e-16
numIntlTrans       3.414e-02  5.318e-03  6.420 1.36e-10
creditLine         1.042e-01  4.705e-03  22.153 2.22e-16

Signif. codes:  0 ‘*’ 0.001 ‘*’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Condition number of final variance-covariance matrix: 3997.308
Number of iterations: 15

```

Next step

[Score new data](#)

Previous step

[Visualize SQL Server data using R](#)

Score new data (SQL and R deep dive)

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

In this step, you use the logistic regression model that you created earlier, to create scores for another data set that uses the same independent variables as inputs.

NOTE

You need DDL admin privileges for some of these steps.

Generate and save scores

1. Update the data source that you set up earlier, `sqlScoreDS`, to add the required column information.

```
sqlScoreDS <- RxSqlServerData(  
  connectionString = sqlConnString,  
  table = sqlScoreTable,  
  colInfo = ccColInfo,  
  rowsPerRead = sqlRowsPerRead)
```

2. To make sure you don't lose the results, create a new data source object. Then, use the new data source object to populate a new table in the SQL Server database.

```
sqlServerOutDS <- RxSqlServerData(table = "ccScoreOutput",  
  connectionString = sqlConnString,  
  rowsPerRead = sqlRowsPerRead )
```

At this point, the table has not been created. This statement just defines a container for the data.

3. Check the current compute context, and set the compute context to the server if needed.

```
rxSetComputeContext(sqlCompute)
```

4. Before running the prediction function that generates the results, you need to check for the existence of an existing output table. Otherwise, you would get an error when you tried to write the new table.

To do this, make a call to the functions **rxSqlServerTableExists** and **rxSqlServerDropTable**, passing the table name as input.

```
if (rxSqlServerTableExists("ccScoreOutput"))      rxSqlServerDropTable("ccScoreOutput")
```

- The function **rxSqlServerTableExists** queries the ODBC driver and returns TRUE if the table exists, FALSE otherwise.
- The function **rxSqlServerDropTable** executes the DDL and returns TRUE if the table is successfully dropped, FALSE otherwise.

- Reference for both functions can be found here: [rxSqlServerDropTable](#)
5. Now you are ready to use the `rxPredict` function to create the scores, and save them in the new table defined in data source `sqlScoreDS`.

```
rxPredict(modelObject = logitObj,
          data = sqlScoreDS,
          outData = sqlServerOutDS,
          predVarNames = "ccFraudLogitScore",
          type = "link",
          writeModelVars = TRUE,
          overwrite = TRUE)
```

The `rxPredict` function is another function that supports running in remote compute contexts. You can use the `rxPredict` function to create scores from models created using `rxLinMod`, `rxLogit`, or `rxGlm`.

- The parameter `writeModelVars` is set to **TRUE** here. This means that the variables that were used for estimation will be included in the new table.
- The parameter `predVarNames` specifies the variable where results will be stored. Here you are passing a new variable, `ccFraudLogitScore`.
- The `type` parameter for `rxPredict` defines how you want the predictions calculated. Specify the keyword **response** to generate scores based on the scale of the response variable. Or, use the keyword **link** to generate scores based on the underlying link function, in which case predictions are created using a logistic scale.

6. After a while, you can refresh the list of tables in Management Studio to see the new table and its data.
7. To add additional variables to the output predictions, use the `extraVarsToWrite` argument. For example, in the following code, the variable `custID` is added from the scoring data table into the output table of predictions.

```
rxPredict(modelObject = logitObj,
          data = sqlScoreDS,
          outData = sqlServerOutDS,
          predVarNames = "ccFraudLogitScore",
          type = "link",
          writeModelVars = TRUE,
          extraVarsToWrite = "custID",
          overwrite = TRUE)
```

Display scores in a histogram

After the new table has been created, you can compute and display a histogram of the 10,000 predicted scores. Computation is faster if you specify the low and high values, so get those from the database and add them to your working data.

1. Create a new data source, `sqlMinMax`, that queries the database to get the low and high values.

```
sqlMinMax <- RxSqlServerData(
  sqlQuery = paste("SELECT MIN(ccFraudLogitScore) AS minValue,", 
                  "MAX(ccFraudLogitScore) AS maxValue FROM ccScoreOutput"),
  connectionString = sqlConnString)
```

From this example, you can see how easy it is to use **RxSqlServerData** data source objects to define arbitrary datasets based on SQL queries, functions, or stored procedures, and then use those in your R code. The variable does not store the actual values, just the data source definition; the query is executed to

generate the values only when you use it in a function like **rxImport**.

2. Call the **rxImport** function to put the values in a data frame that can be shared across compute contexts.

```
minMaxVals <- rxImport(sqlMinMax)
minMaxVals \<- as.vector(unlist(minMaxVals))
```

Results

> *minMaxVals*

[1] -23.970256 9.786345

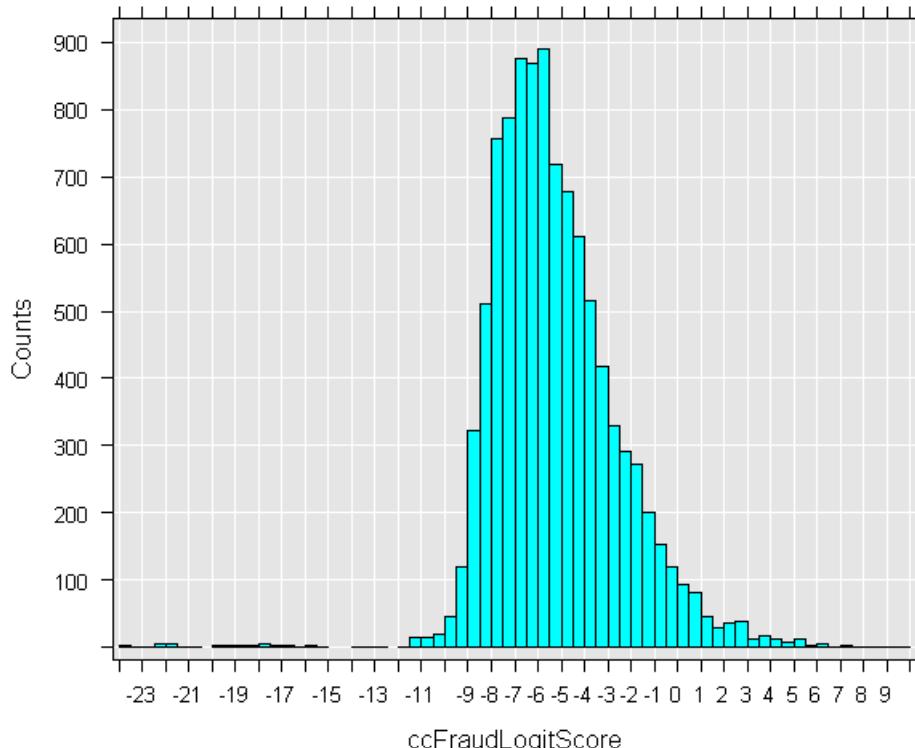
3. Now that the maximum and minimum values are available, use the values to create another data source for the generated scores.

```
sqlOutScoreDS <- RxSqlServerData(sqlQuery = "SELECT ccFraudLogitScore FROM ccScoreOutput",
  connectionString = sqlConnString,
  rowsPerRead = sqlRowsPerRead,
  colInfo = list(ccFraudLogitScore = list(
    low = floor(minMaxVals[1]),
    high = ceiling(minMaxVals[2]) ) ))
```

4. Use the data source object `sqlOutScoreDS` to get the scores, and compute and display a histogram. Add the code to set the compute context if needed.

```
# rxSetComputeContext(sqlCompute)
rxHistogram(~ccFraudLogitScore, data = sqlOutScoreDS)
```

Results



Next step

[Transform data using R](#)

[Previous step](#)

[Create models](#)

Transform data using R (SQL and R deep dive)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

The **RevoScaleR** package provides multiple functions for transforming data at various stages of your analysis:

- **rxDataStep** can be used to create and transform subsets of data.
- **rxImport** supports transforming data as data is being imported to or from an XDF file or an in-memory data frame.
- Although not specifically for data movement, the functions **rxSummary**, **rxCube**, **rxLinMod**, and **rxLogit** all support data transformations.

In this section, you learn how to use these functions. Let's start with [rxDataStep](#).

Use rxDataStep to transform variables

The **rxDataStep** function processes data one chunk at a time, reading from one data source and writing to another. You can specify the columns to transform, the transformations to load, and so forth.

To make this example interesting, let's use a function from another R package to transform the data. The **boot** package is one of the "recommended" packages, meaning that **boot** is included with every distribution of R, but is not loaded automatically on start-up. Therefore, the package should already be available on the SQL Server instance that you've been using with R Services (In-Database).

From the **boot** package, use the function `inv.logit`, which computes the inverse of a logit. That is, the `inv.logit` function converts a logit back to a probability on the [0,1] scale.

TIP

Another way to get predictions in this scale would be to set the `type` parameter to **response** in the original call to `rxPredict`.

1. Start by creating a data source to hold the data destined for the table, `ccScoreOutput`.

```
sqlOutScoreDS <- RxSqlServerData( table = "ccScoreOutput", connectionString = sqlConnString,  
rowsPerRead = sqlRowsPerRead )
```

2. Add another data source to hold the data for the table `ccScoreOutput2`.

```
sqlOutScoreDS2 <- RxSqlServerData( table = "ccScoreOutput2", connectionString = sqlConnString,  
rowsPerRead = sqlRowsPerRead )
```

In the new table, store all the variables from the previous `ccScoreOutput` table, plus the newly created variable.

3. Set the compute context to the SQL Server instance.

```
rxSetComputeContext(sqlCompute)
```

4. Use the function **rxSqlServerTableExists** to check whether the output table `ccScoreOutput2` already exists; and if so, use the function **rxSqlServerDropTable** to delete the table.

```
if (rxSqlServerTableExists("ccScoreOutput2"))      rxSqlServerDropTable("ccScoreOutput2")
```

5. Call the **rxDataStep** function, and specify the desired transforms in a list.

```
rxDataStep(inData = sqlOutScoreDS,
           outFile = sqlOutScoreDS2,
           transforms = list(ccFraudProb = inv.logit(ccFraudLogitScore)),
           transformPackages = "boot",
           overwrite = TRUE)
```

When you define the transformations that are applied to each column, you can also specify any additional R packages that are needed to perform the transformations. For more information about the types of transformations that you can perform, see [How to transform and subset data using RevoScaleR](#).

6. Call **rxGetVarInfo** to view a summary of the variables in the new data set.

```
rxGetVarInfo(sqlOutScoreDS2)
```

Results

Var 1: ccFraudLogitScore, Type: numeric

Var 2: state, Type: character

Var 3: gender, Type: character

Var 4: cardholder, Type: character

Var 5: balance, Type: integer

Var 6: numTrans, Type: integer

Var 7: numIntlTrans, Type: integer

Var 8: creditLine, Type: integer

Var 9: ccFraudProb, Type: numeric

The original logit scores are preserved, but a new column, *ccFraudProb*, has been added, in which the logit scores are represented as values between 0 and 1.

Notice that the factor variables have been written to the table `ccScoreOutput2` as character data. To use them as factors in subsequent analyses, use the parameter *colInfo* to specify the levels.

Next step

[Load data into memory using rxImport](#)

Previous step

[Create and run R scripts](#)

Load data into memory using rxImport (SQL and R deep dive)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

The [rxImport](#) function can be used to move data from a data source into a data frame in session memory, or into an XDF file on disk. If you don't specify a file as destination, data is put into memory as a data frame.

In this step, you learn how to get data from SQL Server, and then use the [rxImport](#) function to put the data of interest into a local file. That way, you can analyze it in the local compute context repeatedly, without having to re-query the database.

Extract a subset of data from SQL Server to local memory

You've decided that you want to examine only the high risk individuals in more detail. The source table in SQL Server is big, so you want to get the information about just the high-risk customers. You then load that data into a data frame in the memory of the local workstation.

1. Reset the compute context to your local workstation.

```
rxSetComputeContext("local")
```

2. Create a new SQL Server data source object, providing a valid SQL statement in the `sqlQuery` parameter.

This example gets a subset of the observations with the highest risk scores. That way, only the data you really need is put in local memory.

```
sqlServerProbDS <- RxSqlServerData(  
  sqlQuery = paste("SELECT * FROM ccScoreOutput2",  
    "WHERE (ccFraudProb > .99)"),  
  connectionString = sqlConnString)
```

3. Call the function [rxImport](#) to read the data into a data frame in the local R session.

```
highRisk <- rxImport(sqlServerProbDS)
```

If the operation was successful, you should see a status message like this one: "Rows Read: 35, Total Rows Processed: 35, Total Chunk Time: 0.036 seconds"

4. Now that the high-risk observations are in an in-memory data frame, you can use various R functions to manipulate the data frame. For example, you can order customers by their risk score, and print a list of the customers who pose the highest risk.

```
orderedHighRisk <- highRisk[order(-highRisk$ccFraudProb),]  
row.names(orderedHighRisk) <- NULL  
head(orderedHighRisk)
```

Results

```
ccFraudLogitScore state gender cardholder balance numTrans numIntlTrans creditLine ccFraudProb 1  
9.786345 SD Male Principal 23456 25 5 75 0.99994382  
9.433040 FL Female Principal 20629 24 28 75 0.99992003  
8.556785 NY Female Principal 19064 82 53 43 0.99980784  
8.188668 AZ Female Principal 19948 29 0 75 0.99972235  
7.551699 NY Female Principal 11051 95 0 75 0.99947516  
7.335080 NV Male Principal 21566 4 6 75 0.9993482
```

More about rxImport

You can use **rxImport** not just to move data, but to transform data in the process of reading it. For example, you can specify the number of characters for fixed-width columns, provide a description of the variables, set levels for factor columns, and even create new levels to use after importing.

The **rxImport** function assigns variable names to the columns during the import process, but you can indicate new variable names by using the *colInfo* parameter, or change data types using the *colClasses* parameter.

By specifying additional operations in the *transforms* parameter, you can do elementary processing on each chunk of data that is read.

Next step

[Create new SQL Server table using rxDataStep](#)

Previous step

[Transform data using R](#)

Create new SQL Server table using rxDataStep (SQL and R deep dive)

4/11/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

In this lesson, you learn how to move data between in-memory data frames, the SQL Server context, and local files.

NOTE

This lesson uses a different data set. The Airline Delays dataset is a public dataset that is widely used for machine learning experiments. The data files used in this example are available in the same directory as other product samples.

Create SQL Server table from local data

In the first half of this tutorial, you used the **RxTextData** function to import data into R from a text file, and then used the **RxDataStep** function to move the data into SQL Server.

This lesson takes a different approach, and uses data from a file saved in the [XDF format](#). After doing some lightweight transformations on the data using the XDF file, you save the transformed data into a new SQL Server table.

What is XDF?

The XDF format is an XML standard developed for high-dimensional data and is the native file format used by [Machine Learning Server](#). It is a binary file format with an R interface that optimizes row and column processing and analysis. You can use it for moving data and to store subsets of data that are useful for analysis.

1. Set the compute context to the local workstation. **DDL permissions are needed for this step.**

```
```R  
rxSetComputeContext("local")
```
```

1. Define a new data source object using the **RxXdfData** function. To define an XDF data source, specify the path to the data file.

You could specify the path to the file using a text variable. However, in this case, there's a handy shortcut, which is to use the **rxGetOption** function and get the file (AirlineDemoSmall.xdf) from the sample data directory.

```
xdfAirDemo <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.xdf"))
```

2. Call **rxGetVarInfo** on the in-memory data to view a summary of the dataset.

```
rxGetVarInfo(xdfAirDemo)
```

Results

Var 1: *ArrDelay*, Type: *integer*, Low/High: (-86, 1490)

Var 2: *CRSDepTime*, Type: *numeric*, Storage: *float32*, Low/High: (0.0167, 23.9833)

Var 3: *DayOfWeek* 7 factor levels: Monday Tuesday Wednesday Thursday Friday Saturday Sunday

NOTE

Did you notice that you did not need to call any other functions to load the data into the XDF file, and could call **rxGetVarInfo** on the data immediately? That's because XDF is the default interim storage method for RevoScaleR. In addition to XDF files, the **rxGetVarInfo** function now supports multiple source types.

1. Put this data into a SQL Server table, storing *DayOfWeek* as an integer with values from 1 to 7.

To do this, first define a SQL Server data source.

```
sqlServerAirDemo <- RxSqlServerData(table = "AirDemoSmallTest", connectionString = sqlConnString)
```

2. Check whether a table with the same name already exists, and delete the table if it exists.

```
if (rxSqlServerTableExists("AirDemoSmallTest", connectionString = sqlConnString))
  rxSqlServerDropTable("AirDemoSmallTest", connectionString = sqlConnString)
```

3. Create the table and load the data using **rxDataStep**. This function moves data between two already defined data sources and can optionally transform the data en route.

```
rxDataStep(inData = xdfAirDemo, outFile = sqlServerAirDemo,
           transforms = list( DayOfWeek = as.integer(DayOfWeek),
                             rowNum = .rxStartRow : (.rxStartRow + .rxNumRows - 1) ),
           overwrite = TRUE )
```

This is a fairly large table, so wait until you see a final status message like this one: *Rows Read: 200000, Total Rows Processed: 600000*.

4. Set the compute context back to the SQL Server computer.

```
rxSetComputeContext(sqlCompute)
```

5. Create a new SQL Server data source, using a simple SQL query on the new table. This definition adds factor levels for the *DayOfWeek* column, using the *colInfo* argument to **RxSqlServerData**.

```
SqlServerAirDemo <- RxSqlServerData(
  sqlQuery = "SELECT * FROM AirDemoSmallTest",
  connectionString = sqlConnString,
  rowsPerRead = 50000,
  colInfo = list(DayOfWeek = list(type = "factor", levels = as.character(1:7))))
```

6. Call **rxSummary** once more to review a summary of the data in your query.

```
rxSummary(~., data = sqlServerAirDemo)
```

Next step

[Perform chunking analysis using rxDataStep](#)

Previous step

[Load data into memory using rxImport](#)

Perform chunking analysis using rxDataStep (SQL and R deep dive)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

In this lesson, you use the **rxDataStep** function to process data in chunks, rather than requiring that the entire dataset be loaded into memory and processed at one time, as in traditional R. The **rxDataStep** functions reads the data in chunk, applies R functions to each chunk of data in turn, and then saves the summary results for each chunk to a common SQL Server data source. When all data has been read, the results are combined.

TIP

For this lesson, you compute a contingency table by using the `table` function in R. This example is meant for instructional purposes only.

If you need to tabulate real-world data sets, we recommend that you use the **rxCrossTabs** or **rxCube** functions in [RevoScaleR](#), which are optimized for this sort of operation.

Partition data by values

1. Create a custom R function that calls the R `table` function on each chunk of data, and name the new function `ProcessChunk`.

```
ProcessChunk <- function( dataList ) {  
  # Convert the input list to a data frame and compute contingency table  
  chunkTable <- table(as.data.frame(dataList))  
  
  # Convert table output to a data frame with a single row  
  varNames <- names(chunkTable)  
  varValues <- as.vector(chunkTable)  
  dim(varValues) <- c(1, length(varNames))  
  chunkDF <- as.data.frame(varValues)  
  names(chunkDF) <- varNames  
  
  # Return the data frame, which has a single row  
  return( chunkDF )  
}
```

2. Set the compute context to the server.

```
rxSetComputeContext( sqlCompute )
```

3. Define a SQL Server data source to hold the data you're processing. Start by assigning a SQL query to a variable. Then, use that variable in the `sqlQuery` argument of a new SQL Server data source.

```
```R
dayQuery <- "SELECT DayOfWeek FROM AirDemoSmallTest"
inDataSource <- RxSqlServerData(sqlQuery = dayQuery,
 connectionString = sqlConnString,
 rowsPerRead = 50000,
 colInfo = list(DayOfWeek = list(type = "factor",
 levels = as.character(1:7))))
```

```

1. Optionally, you can run **rxGetVarInfo** on this data source. At this point, it contains a single column: *Var 1: DayOfWeek, Type: factor, no factor levels available*
2. Before applying this factor variable to the source data, create a separate table to hold the intermediate results. Again, you just use the RxSqlServerData function to define the data, making sure to delete any existing tables of the same name.

```
iroDataSource = RxSqlServerData(table = "iroResults", connectionString = sqlConnString)
# Check whether the table already exists.
if (rxSqlServerTableExists(table = "iroResults", connectionString = sqlConnString)) {
  rxSqlServerDropTable( table = "iroResults", connectionString = sqlConnString) }
```

3. Call the custom function **ProcessChunk** to transform the data as it is read, by using it as the *transformFunc* argument to the **rxDataStep** function.

```
rxDataStep( inData = inDataSource, outFile = iroDataSource, transformFunc = ProcessChunk, overwrite =
TRUE)
```

4. To view the intermediate results of **ProcessChunk**, assign the results of **rxImport** to a variable, and then output the results to the console.

```
iroResults <- rxImport(iroDataSource)
iroResults
```

Partial results

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|------|------|------|------|------|------|------|
| 1 | 8228 | 8924 | 6916 | 6932 | 6944 | 5602 | 6454 |
| 2 | 8321 | 5351 | 7329 | 7411 | 7409 | 6487 | 7692 |

1. To compute the final results across all chunks, sum the columns, and display the results in the console.

```
finalResults <- colSums(iroResults)
finalResults
```

Results

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-------|-------|-------|-------|-------|-------|
| 97975 | 77725 | 78875 | 81304 | 82987 | 86159 | 94975 |

2. To remove the intermediate results table, make a call to **rxSqlServerDropTable**.

```
rxSqlServerDropTable( table = "iroResults", connectionString = sqlConnString)
```

Next step

[Analyze data in local compute context](#)

Previous step

[Create new SQL Server table using rxDataStep](#)

Analyze data in local compute context (SQL and R deep dive)

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

In this section, you learn how to switch back to a local compute context, and move data between contexts to optimize performance.

Although it might be faster to run complex R code using the server context, sometimes it is more convenient to get your data out of SQL Server and analyze it on a local workstation.

Create a local summary

1. Change the compute context to do all your work locally.

```
rxSetComputeContext ("local")
```

2. When extracting data from SQL Server, you can often get better performance by increasing the number of rows extracted for each read. To do this, increase the value for the *rowsPerRead* parameter on the data source. Previously, the value of *rowsPerRead* was set to 5000.

```
sqlServerDS1 <- RxSqlServerData(  
  connectionString = sqlConnString,  
  table = sqlFraudTable,  
  colInfo = ccColInfo,  
  rowsPerRead = 10000)
```

3. Call **rxSummary** on the new data source.

```
rxSummary(formula = ~gender + balance + numTrans + numIntlTrans + creditLine, data = sqlServerDS1)
```

The actual results should be the same as when you run **rxSummary** in the context of the SQL Server computer. However, the operation might be faster or slower. Much depends on the connection to your database, because the data is being transferred to your local computer for analysis.

Next step

[Move data between SQL Server and XDF File](#)

Previous step

[Perform chunking analysis using rxDataStep](#)

Move data between SQL Server and XDF file (SQL and R deep dive)

4/11/2018 • 3 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is part of the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

In this step, you learn to use an XDF file to transfer data between remote and local compute contexts. Storing the data in an XDF file allows you to perform transformations on the data.

When you're done, you use the data in the file to create a new SQL Server table. The function [rxDataStep](#) can apply transformations to the data and performs the conversion between data frames and .xdf files.

Create a SQL Server table from an XDF file

For this exercise, you use the credit card fraud data again. In this scenario, you've been asked to do some extra analysis on users in the states of California, Oregon, and Washington. To be more efficient, you've decided to store data for only these states on your local computer, and work with only the variables gender, cardholder, state, and balance.

1. Re-use the `stateAbb` variable you created earlier to identify the levels to include, and write them to a new variable, `statesToKeep`.

```
statesToKeep <- sapply(c("CA", "OR", "WA"), grep, stateAbb)
statesToKeep
```

Results

| CA | OR | WA |
|----|----|----|
| 5 | 38 | 48 |

2. Define the data you want to bring over from SQL Server, using a Transact-SQL query. Later you use this variable as the `inData` argument for [rxImport](#).

```
importQuery <- paste("SELECT gender,cardholder,balance,state FROM", sqlFraudTable, "WHERE (state = 5
OR state = 38 OR state = 48)")
```

Make sure there are no hidden characters such as line feeds or tabs in the query.

3. Next, define the columns to use when working with the data in R. For example, in the smaller data set, you need only three factor levels, because the query returns data for only three states. Apply the `statesToKeep` variable to identify the correct levels to include.

```
importColInfo <- list(
  gender = list( type = "factor", levels = c("1", "2"), newLevels = c("Male", "Female")),
  cardholder = list( type = "factor", levels = c("1", "2"), newLevels = c("Principal",
  "Secondary")),
  state = list( type = "factor", levels = as.character(statesToKeep), newLevels =
  names(statesToKeep))
)
```

- Set the compute context to **local**, because you want all the data available on your local computer.

```
rxSetComputeContext("local")
```

The **rxImport** function can import data from any supported data source to a local XDF file. Using a local copy of the data is convenient when you want to do many different analyses on the data, but want to avoid running the same query over and over.

- Create the data source object by passing the variables previously defined as arguments to **RxSqlServerData**.

```
sqlServerImportDS <- RxSqlServerData(
  connectionString = sqlConnString,
  sqlQuery = importQuery,
  colInfo = importColInfo)
```

- Call **rxImport** to write the data to a file named `ccFraudSub.xdf`, in the current working directory.

```
localDS <- rxImport(inData = sqlServerImportDS,
  outFile = "ccFraudSub.xdf",
  overwrite = TRUE)
```

The `localDS` object returned by the **rxImport** function is a light-weight **RxXdfData** data source object that represents the `ccFraud.xdf` data file stored locally on disk.

- Call **rxGetVarInfo** on the XDF file to verify that the data schema is the same.

```
rxGetVarInfo(data = localDS)
```

Results

```
rxGetVarInfo(data = localDS)
```

Var 1: gender, Type: factor, no factor levels available

Var 2: cardholder, Type: factor, no factor levels available

Var 3: balance, Type: integer, Low/High: (0, 22463)

Var 4: state, Type: factor, no factor levels available

- You can now call various R functions to analyze the `localDS` object, just as you would with the source data on SQL Server. For example, you might summarize by gender:

```
rxSummary(~gender + cardholder + balance + state, data = localDS)
```

Now that you've mastered the use of compute contexts and working with various data sources, it's time to try

something fun. In the next and final lesson, you create a simple simulation that runs a custom R function on the remote server.

Next step

[Create a simple simulation](#)

Previous step

[Analyze data in local compute context](#)

Create a simple simulation (SQL and R deep dive)

4/11/2018 • 4 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is the final step in the Data Science Deep Dive tutorial, on how to use [RevoScaleR](#) with SQL Server.

Until now you've been using R functions that are designed specifically for moving data between SQL Server and a local compute context. However, suppose you write a custom R function of your own, and want to run it in the server context?

You can call an arbitrary function in the context of the SQL Server computer, by using the `rxExec` function. You can also use `rxExec` to explicitly distribute work across cores in a single server.

In this lesson, you use the remote server to create a simple simulation. The simulation doesn't require any SQL Server data; the example only demonstrates how to design a custom function and then call it using the `rxExec` function.

For a more complex example of using `rxExec`, see this article: [Coarse grain parallelism with foreach and rxExec](#)

Create the custom function

A common casino game consists of rolling a pair of dice, with these rules:

- If you roll a 7 or 11 on your initial roll, you win.
- If you roll 2, 3, or 12, you lose.
- If you roll a 4, 5, 6, 8, 9, or 10, that number becomes your point and you continue rolling until you either roll your point again (in which case you win) or roll a 7, in which case you lose.

The game is easily simulated in R, by creating a custom function, and then running it many times.

1. Create the custom function using the following R code:

```

rollDice <- function()
{
  result <- NULL
  point <- NULL
  count <- 1
  while (is.null(result))
  {
    roll <- sum(sample(6, 2, replace=TRUE))

    if (is.null(point))
    { point <- roll }
    if (count == 1 && (roll == 7 || roll == 11))
    { result <- "Win" }
    else if (count == 1 && (roll == 2 || roll == 3 || roll == 12))
    { result <- "Loss" }
    else if (count > 1 && roll == 7 )
    { result <- "Loss" }
    else if (count > 1 && point == roll)
    { result <- "Win" }
    else { count <- count + 1 }
  }
  result
}

```

2. To simulate a single game of dice, run the function.

```
rollDice()
```

Did you win or lose?

Now let's see how you can use **rxExec** to run the function multiple times, to create a simulation that helps determine the probability of a win.

Create the simulation

To run an arbitrary function in the context of the SQL Server computer, you call the **rxExec** function. Although **rxExec** also supports distributed execution of a function in parallel across nodes or cores in a server context, here it runs your custom function on the SQL Server computer.

1. Call the custom function as an argument to **rxExec**, together with other parameters that modify the simulation.

```
sqlServerExec <- rxExec(rollDice, timesToRun=20, RNGseed="auto")
length(sqlServerExec)
```

- Use the *timesToRun* argument to indicate how many times the function should be executed. In this case, you roll the dice 20 times.
- The arguments *RNGseed* and *RNGkind* can be used to control random number generation. When *RNGseed* is set to **auto**, a parallel random number stream is initialized on each worker.

2. The **rxExec** function creates a list with one element for each run; however, you won't see much happening until the list is complete. When all the iterations are complete, the line starting with `length` will return a value.

You can then go to the next step to get a summary of your win-loss record.

3. Convert the returned list to a vector using R's `unlist` function, and summarize the results using the `table`

function.

```
table(unlist(sqlServerExec))
```

Your results should look something like this:

Loss Win 12 8

Conclusions

In this tutorial, you have become proficient with these tasks:

- Getting SQL Server data to use in analyses
- Creating and modifying data sources in R
- Passing models, data, and plots between your workstation and the SQL Server server

If you would like to experiment with these techniques using a larger dataset of 10 million observations, the data files are available from the Revolution analytics web site: [Index of datasets](#)

To re-use this walkthrough with the larger data files, download the data, and then modify each of the data sources as follows:

1. Modify the variables `ccFraudCsv` and `ccScoreCsv` to point to the new data files
2. Change the name of the table referenced in `sqlFraudTable` to `ccFraud10`
3. Change the name of the table referenced in `sqlScoreTable` to `ccFraudScore10`

Additional samples

Now that you've mastered the use of compute contexts and RevoScaler functions to pass and transform data, check out these tutorials:

[R tutorials for Machine Learning Services](#)

Previous step

[Move data between SQL Server and XDF file](#)

SQL Server Python tutorials

4/11/2018 • 2 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides a list of tutorials and samples that demonstrate the use of Python with SQL Server 2017. Through these samples and demos, you will learn:

- How to run Python from T-SQL
- What are remote and local compute contexts, and how you can execute Python code using the SQL Server computer
- How to wrap Python code in a stored procedure
- Optimizing Python code for a SQL production environment
- Real-world scenarios for embedding machine learning in applications

For information about requirements and setup, see [Prerequisites](#).

Python tutorials

- [Running Python in T-SQL](#)

Learn the basics of how to call Python in T-SQL, using the extensibility mechanism pioneered in SQL Server 2016.

- [Create a machine learning model in Python using revoscalepy](#)

This lesson demonstrates how you can run code from a remote Python terminal, using SQL Server compute context. You should be somewhat familiar with Python tools and environments. Sample code is provided that creates a model using **rxLinMod**, from the new **revoscalepy** library.

- [In-Database Python analytics for SQL developers](#)

This end-to-end walkthrough demonstrates the process of building a complete Python solution using T-SQL stored procedures. All Python code is included.

Python samples

These samples and demos provided by the SQL Server development team highlight ways that you can use embedded analytics in real-world applications.

- [Build a predictive model using Python and SQL Server](#)

Learn how a ski rental business might use machine learning to predict future rentals, which helps the business plan and staff to meet future demand.

TIP

Now includes native scoring from Python models!

- [Perform customer clustering using Python and SQL Server](#)

Learn how to use the Kmeans algorithm to perform unsupervised clustering of customers.

Prerequisites

To use these tutorials, you must have SQL Server 2017, and you must explicitly install and then enable the feature, Machine Learning Services (In-Database).

SQL Server 2017 supports both the R and Python languages, but neither is installed or enabled by default. Running Python requires that the extensibility framework be enabled, and that you select Python as the language to install.

Post-installation configuration tips

After running SQL Server setup, you might need to perform some additional steps to ensure that Python and SQL Server are communicating:

- Enable the external script execution feature by running `sp_configure 'external scripts enabled', 1`.
- Restart the server.
- Open the **Services** panel to check whether Launchpad has started.
- Ensure that the service that calls the external runtime has necessary permissions. For more information, see [Enable implied authentication](#).
- Open a port on the firewall for SQL Server, and enable required network protocols.
- Ensure that your SQL login or Windows user account has necessary permissions to connect to the server, to read data, and to create any database objects required by the sample.

See this article for some common issues: [Troubleshooting Machine Learning Services](#)

Resource management

You can install both R and Python on the same computer, but running both can require substantial resources. If you get "out of memory" errors, or if running machine learning jobs is the principal intended use of the server, you can reduce the amount of memory that is allocated to the database engine. For more information, see [Managing and monitoring Python in SQL Server](#).

See also

[R tutorials for SQL Server](#)

Run Python using T-SQL

4/11/2018 • 11 minutes to read • [Edit Online](#)

THIS TOPIC APPLIES TO: SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This tutorial explains how you can run Python code in SQL Server 2017. It walks you through the process of moving data between SQL Server and Python, and explains how to wrap well-formed Python code in a stored procedure `sp_execute_external_script` to build, train, and use machine learning models in SQL Server.

Prerequisites

To complete this tutorial, you must first install SQL Server 2017 and enable Machine Learning Services on the instance, as described in [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#).

You should also install [SQL Server Management Studio](#). Alternatively, you can use another database management or query tool, as long as it can connect to a server and database, and run a T-SQL query or stored procedure.

After you have completed setup, return to this tutorial, to learn how to execute Python code in the context of a stored procedure.

Overview

This tutorial includes four lessons:

- The basics of moving data between SQL Server and Python: learn the basic requirements, data structures, inputs, and outputs.
- Practice using stored procedures for simple Python tasks, like loading sample data.
- Use stored procedures to create a Python machine learning model, and generate scores from the model.
- An optional lesson for users who intend to run Python from a remote client, using SQL Server as the *compute context*. Includes code for building a model; however, requires that you are already somewhat familiar with Python environments and Python tools.

Additional Python samples specific to SQL Server 2017 are provided here: [SQL Server Python tutorials](#)

Verify that Python is enabled and the Launchpad is running

1. In Management Studio, run this statement to make sure the service has been enabled.

```
sp_configure 'external scripts enabled'
```

If **run_value** is 1, the machine learning feature is installed and ready to use.

A common cause of errors is that the Launchpad, which manages communication between SQL Server and Python, has stopped. You can view the Launchpad status by using the Windows **Services** panel, or by opening SQL Server Configuration Manager. If the service has stopped, restart it.

2. Next, verify that the Python runtime is working and communicating with SQL Server. To do this, open a new **Query** window in SQL Server Management Studio, and connect to the instance where Python was installed.

```
EXEC sp_execute_external_script @language = N'Python',
@script = N'print(3+4)'
```

If all is well, you should see a result message like this one

```
STDOUT message(s) from external script:
7
```

3. If you get errors, there are a variety of things you can do to ensure that the server and Python can communicate.

You must add the Windows user group `SQLUserGroup` as a login on the instance, to ensure that Launchpad can provide communication between Python and SQL Server. (The same group is used for both R and Python code execution.) For more information, see [Enabled implied authentication](#).

Additionally, you might need to enable network protocols that have been disabled, or open the firewall so that SQL Server can communicate with external clients. For more information, see [Troubleshooting setup](#).

Basic Python interaction

There are two ways to run Python code in SQL Server:

- Add a Python script as an argument of the system stored procedure, `sp_execute_external_script`
- From a remote Python client, connect to SQL Server, and execute code using the SQL Server as the compute context. This requires [revoscalepy](#).

The primary goal of this tutorial is to ensure that you can use Python in a stored procedure.

1. Run some simple code to see how data is passed back and forth between SQL Server and Python.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
a = 1
b = 2
c = a/b
d = a*b
print(c, d)
'
```

2. Assuming that you have everything set up correctly, and Python and SQL Server are talking to each other, the correct result is calculated, and the Python `print` function returns the result to the **Messages** windows.

Results

```
STDOUT message(s) from external script:
0.5 2
```

While getting **stdout** messages is handy when testing your code, more often you need to return the results in tabular format, so that you can use it in an application or write it to a table.

For now, remember these rules:

- Everything inside the `@script` argument must be valid Python code.
- The code must follow all Pythonic rules regarding indentation, variable names, and so forth. When you get an

error, check your white space and casing.

- If you are using any libraries that are not loaded by default, you must use an import statement at the beginning of your script to load them.
- If the library is not already installed, stop, and install the Python package outside of SQL Server, as described here: [Install new Python packages on SQL Server](#)

Inputs and outputs

By default, `sp_execute_external_script` accepts a single input dataset, which typically you supply in the form of a valid SQL query. Other types of input can be passed as SQL variables: for example, you can pass a trained model as a variable, using a serialization function such as `pickle` or `rx_serialize_model` to write the model in a binary format.

The stored procedure returns a single Python `pandas` data frame as output. However you can output scalars and models as variables. For example, you can output a trained model as a binary variable and pass that to a T-SQL `INSERT` statement, to write that model to a table. You can also generate plots (in binary format) or scalars (individual values, such as the date and time, the time elapsed to train the model, and so forth).

For now, let's look at just the default input and output variables, `InputDataSet` and `OutputDataSet`.

1. Run the following code to do some math and output the results.

```
```sql
execute sp_execute_external_script
@language = N'Python',
@script = N'
a = 1
b = 2
c = a/b
print(c)
OutputDataSet = c
'

WITH RESULT SETS ((ResultValue float))
```
```

2. You should get an error, because the Python code generates a scalar, not a data frame.

```
**Results**

```text
line 43, in transform
 raise TypeError('OutputDataSet should be of type pandas.DataFrame')
```
```
```

3. Now see what happens when you pass a tabular dataset to Python, using the default input variable `InputDataSet`.

```
EXECUTE sp_execute_external_script
@language = N'Python',
@script = N'
OutputDataSet = InputDataSet
',
@input_data_1 = N'SELECT 1 as Col1'
```

The stored procedure returns a `data.frame` automatically, without you having to do anything extra in your Python code.

## Results

NO COLUMNNNAME

1

By default, the single tabular input dataset has the name, `InputDataSet`. However, you can change that name by adding a line like this: `@input_data_1_name = N'myResultName'`.

Column names used by Python are never preserved in the output. Although the input query specified the column name `Col1`, that name is not returned, nor would any column headings used by your Python script. To specify a column name and data type when you return the data to SQL Server, use the T-SQL `WITH RESULT SETS` clause.

4. This example provides new names for the input and output variables.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
MyOutput = MyInput
',
@input_data_1_name = N'MyInput',
@input_data_1 = N'SELECT 1 as Col1',
@output_data_1_name = N'MyOutput'
WITH RESULT SETS ((ResultValue int))
```

The `WITH RESULT SET` clause defines the schema for the output, since Python column names are never returned with the data.frame.

## Results

RESULTVALUE

1

5. Now let's look at a typical Python error. Change the line in the previous example from

`@input_data_1_name = N'MyInput'` to `@input_data_1_name = N'myinput'`.

Python errors are passed to you as messages, by the satellite service used by SQL Server. Messages can be long and include SQL Server errors or Launchpad errors in addition to Python errors, so be patient in digging through the text. The key message is in this line:

```
MyOutput = MyInput
NameError: name 'MyInput' is not defined
```

Recall that Python, like R, is case-sensitive. Therefore, when you get any kind of error, be sure to check your variable names, and look for issues with spacing, indentation, and data types.

## Python data structures

SQL Server relies on the Python **pandas** package, which is great for working with tabular data. However, you've already seen that you cannot pass a scalar from Python to SQL Server and expect it to "just work". In this section, we'll review some basic data type definitions, to prepare you for additional issues that you might run across when passing tabular data between Python and SQL Server.

- A data frame is a table with *multiple* columns.

- A single column of a DataFrame, is a list-like object called a Series.
- A single value is a cell of a data frame and has to be called by index.

So how would you expose the single result of a calculation as a data frame, if a data.frame requires a tabular structure? One answer is to represent the single scalar value as a series, which is easily converted to a data frame.

1. This example does some simple math and converts a scalar into a series. A series requires an index, which you can assign manually, as shown here, or programmatically.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
a = 1
b = 2
c = a/b
print(c)
s = pandas.Series(c, index =["simple math example 1"])
print(s)
'
```

2. Because the series hasn't been converted to a data.frame, the values are returned in the Messages window, but you can see that the results are in a more tabular format.

## Results

```
STDOUT message(s) from external script:
0.5
simple math example 1 0.5
dtype: float64
```

3. To increase the length of the series, you can add new values, using an array.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
a = 1
b = 2
c = a/b
d = a*b
s = pandas.Series([c,d])
print(s)
'
```

If you do not specify an index, an index is generated that has values starting with 0 and ending with the length of the array.

## Results

```
STDOUT message(s) from external script:
0 0.5
1 2.0
dtype: float64
```

4. If you increase the number of **index** values, but don't add new **data** values, the data values are repeated to fill the series.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
a = 1
b = 2
c = a/b
s = pandas.Series(c, index =["simple math example 1", "simple math example 2"])
print(s)
'
```

## Results

```
STDOUT message(s) from external script:
0.5
simple math example 1 0.5
simple math example 2 0.5
dtype: float64
```

## Convert series to data frame

Having converted our scalar math results to a tabular structure, we still need to convert them to a format that SQL Server can handle.

1. To convert a series to a data.frame, call the pandas [DataFrame](#) method.

```
execute sp_execute_external_script
@language = N'Python',
@script = N'
import pandas as pd
a = 1
b = 2
c = a/b
d = a*b
s = pandas.Series([c,d])
print(s)
df = pd.DataFrame(s)
OutputDataSet = df
'

WITH RESULT SETS ((ResultValue float))
```

2. Note that the index values aren't output, even if you use the index to get specific values from the data.frame.

## Results

RESULTVALUE
0.5
2

## Output values into data.frame using an index

Let's see how conversion to a data.frame works with our two series containing the results of simple math operations. The first has an index of sequential values generated by Python. The second uses an arbitrary index of string values.

1. This example gets a value from the series that uses an integer index.

```

EXECUTE sp_execute_external_script
@language = N'Python',
@script = N'
import pandas as pd
a = 1
b = 2
c = a/b
d = a*b
s = pandas.Series([c,d])
print(s)
df = pd.DataFrame(s, index=[1])
OutputDataSet = df
'

WITH RESULT SETS ((ResultValue float))

```

Remember that the auto-generated index starts at 0. Try using an out of range index value and see what happens.

- Now let's get a single value from the other data frame that has a string index.

```

EXECUTE sp_execute_external_script
@language = N'Python',
@script = N'
import pandas as pd
a = 1
b = 2
c = a/b
s = pandas.Series(c, index =["simple math example 1", "simple math example 2"])
print(s)
df = pd.DataFrame(s, index=["simple math example 1"])
OutputDataSet = df
'

WITH RESULT SETS ((ResultValue float))

```

## Results

RESULTVALUE
0.5

If you try to use a numeric index to get a value from this series, you get an error.

This exercise was intended to give you an idea of how to work with different Python data structures and ensure you get the right result as a data frame. You might have concluded that outputting a single value as a data frame is more trouble than its worth! Fortunately, you can easily pass all kinds of values in and out of the stored procedure as variables. That's covered in the next lesson.

## Tips

- Among programming languages, Python is one of the most flexible with regard to single quotes vs. double quotation marks; they're pretty much interchangeable.

However, T-SQL uses single quotes for only certain things, and the `@script` argument uses single quotes to enclose the Python code as a Unicode string. Therefore, you might need to review your Python code and change some single quotes to double quotes.

- Can't find the stored procedure, `sp_execute_external_script`? It means you probably haven't finished configuring the instance to support external script execution. After running SQL Server 2017 setup and selecting Python as the machine learning language, you must also explicitly enable the feature using

`sp_configure`, and then restart the instance.

For details, see [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#).

## Next steps

[Wrap Python code in a SQL stored procedure](#)

# Wrap Python code in a stored procedure

5/3/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:**  SQL Server (Windows only)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In a [previous lesson](#), you learned how to make Python talk to SQL Server. In this lesson, you learn how to embed Python code in a stored procedure, to get data from the Python sample datasets, and write that data to a SQL Server table.

The system stored procedure `sp_execute_external_script` provides the wrapper that passes SQL variables and SQL datasets into Python. It also handles the results output by Python and passes them to SQL Server in a format compatible with SQL data types.

Let's see how this works.

## Prepare the database and tables

Although it is possible to set up a remote client and run Python code using Visual Studio Code, Visual Studio, PyCharm, or other tools, to simplify the scenario, all code in this lesson should be run as part of a stored procedure.

1. Start SQL Server Management Studio, and open a new **Query** window.
2. Create a new database for this project, and change the context of your **Query** window to use the new database.

```
CREATE DATABASE sqlpy
GO
USE sqlpy
GO
```

### TIP

If you're new to SQL Server, or are working on a server you own, a common mistake is to log in and start working without noticing that you are in the **master** database. To be sure that you are using the correct database, always specify the context using the `USE <database name>` statement.

3. Add some empty tables: one to store the data, and one to store the models you train. Later you populate the tables using Python.

The following code creates the table for the training data.

```
DROP TABLE IF EXISTS iris_data;
GO
CREATE TABLE iris_data (
 id INT NOT NULL IDENTITY PRIMARY KEY
 , "Sepal.Length" FLOAT NOT NULL, "Sepal.Width" FLOAT NOT NULL
 , "Petal.Length" FLOAT NOT NULL, "Petal.Width" FLOAT NOT NULL
 , "Species" VARCHAR(100) NOT NULL, "SpeciesId" INT NOT NULL
);
```

If you are new to T-SQL, it pays to memorize the `DROP...IF` statement. When you try to create a table and

one already exists, SQL Server returns an error: "There is already an object named 'iris\_data' in the database." One way to avoid such errors is to delete any existing tables or other objects as part of your code.

4. Run the following code to create the table used for storing the trained model. To save Python (or R) models in SQL Server, they must be serialized and stored in a column of type **varbinary(max)**.

```
DROP TABLE IF EXISTS iris_models;
GO

CREATE TABLE iris_models (
 model_name VARCHAR(50) NOT NULL DEFAULT('default model') PRIMARY KEY,
 model VARBINARY(MAX) NOT NULL
);
GO
```

In addition to the model contents, typically, you would also add columns for other useful metadata, such as the model's name, the date it was trained, the source algorithm and parameters, source data, and so forth. For now we'll keep it simple and use just the model name.

## Populate the table

To move the training data from Python into a SQL Server table is a multistep process:

- You design a stored procedure that gets the data you want.
- You execute the stored procedure to actually get the data.
- You use an INSERT statement to specify where the retrieved data should be saved.

1. Create the following stored procedure that includes Python code.

```
CREATE PROCEDURE get_iris_dataset
AS
BEGIN
 EXEC sp_execute_external_script @language = N'Python',
 @script = N'
from sklearn import datasets
iris = datasets.load_iris()
iris_data = pandas.DataFrame(iris.data)
iris_data["Species"] = pandas.Categorical.from_codes(iris.target, iris.target_names)
iris_data["SpeciesId"] = iris.target
',
 @input_data_1 = N'',
 @output_data_1_name = N'iris_data'
 WITH RESULT SETS (("Sepal.Length" float not null, "Sepal.Width" float not null, "Petal.Length" float not
null, "Petal.Width" float not null, "Species" varchar(100) not null, "SpeciesId" int not null));
END;
GO
```

When you run this code, you should get the message "Commands completed successfully." All this means is that the stored procedure has been created according to your specifications.

2. To actually populate the table, run the stored procedure and specify the table where the data should be written. When run, the stored procedure executes the Python code, which loads the Iris dataset from the built-in Python sample data.

```
INSERT INTO iris_data ("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species",
"SpeciesId")
EXEC dbo.get_iris_dataset;
```

If you're new to T-SQL, be aware that the `INSERT` statement only adds new data; it won't check for existing data, or delete and rebuild the table. To avoid getting multiple copies of the same data in a table, you can run this statement first: `TRUNCATE TABLE iris_data`. The T-SQL `TRUNCATE TABLE` statement deletes existing data but keeps the structure of the table intact.

**TIP**

To modify the stored procedure later, you don't need to drop and recreate it. Use the `ALTER PROCEDURE` statement.

3. To verify that the data was loaded correctly, you can run some simple queries:

```
SELECT TOP(10) * FROM iris_data;
SELECT COUNT(*) FROM iris_data;
```

In the [next lesson](#), you create a machine learning model and save it to a table.

### Further reading about stored procedures

If you are new to SQL Server, you might find stored procedures complicated at first. But a stored procedure is a powerful and flexible interface for passing data between applications and the server. By using a stored procedure, you can dynamically define inputs, which makes it easy to pass in new model names, new parameters, and new data, without altering your Python code.

For an overview of how stored procedures work, see [Stored Procedures \(Database Engine\)](#), or this tutorial: [Writing Transact-SQL Statements](#).

There are also some good tutorials at community sites such as [SQL Server Central](#) or [SQL Team](#).

As you think about how you can best encapsulate Python code in T-SQL, also consider using these features:

- Defining default values for the stored procedure
- Using the `OUTPUT` keyword to pass through input variables
- Creating schema definitions using `WITH RESULTS` to ensure that data consumed by applications has the right data types and column names
- Providing hints to improve batch processing
- Impersonating a different user to test your code, using the `EXECUTE AS` clause

## Next lesson

[Train a Python model and generate scores in SQL Server](#)

# Use Python model in SQL for training and scoring

7/16/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In the [previous lesson](#), you learned the common pattern for using Python together with SQL. You learned that your Python code should output one clearly defined data.frame, and can optionally output multiple scalar or binary variables. You've learned that the SQL stored procedure should be designed to pass the right type of data into Python, and handle the results.

In this section, you use this same pattern to train a model on the data you've added to SQL Server, and save the model to a SQL Server table:

- You design a stored procedure that calls a Python machine learning function.
- The stored procedure needs data from SQL Server to use in training the model.
- The stored procedure outputs a trained model as a binary variable.
- You save the trained model by inserting the variable model into a table.

## Create the stored procedure and train a Python model

1. Run the following code in SQL Server Management Studio to create the stored procedure that builds a model.

```
CREATE PROCEDURE generate_iris_model (@trained_model varbinary(max) OUTPUT)
AS
BEGIN
 EXEC sp_execute_external_script @language = N'Python',
 @script = N'
import pickle
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
trained_model = pickle.dumps(GNB.fit(iris_data[[0,1,2,3]], iris_data[[4]]))
'

 , @input_data_1 = N'select "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "SpeciesId"
from iris_data'
 , @input_data_1_name = N'iris_data'
 , @params = N'@trained_model varbinary(max) OUTPUT'
 , @trained_model = @trained_model OUTPUT;
END;
GO
```

2. If this command runs without error, a new stored procedure is created and added to the database. You can find stored procedures in Management Studio's **Object Explorer**, under **Programmability**.
3. Now run the stored procedure.

```
EXEC generate_iris_model
```

You should get an error, because you haven't provided the input the stored procedure requires.

"Procedure or function 'generate\_iris\_model' expects parameter '@trained\_model', which was not supplied."

4. To generate the model with the required inputs and save it to a table requires some additional statements:

```
DECLARE @model varbinary(max);
EXEC generate_iris_model @model OUTPUT;
INSERT INTO iris_models (model_name, model) values('Naive Bayes', @model);
```

- Now, try running the model generation code once more.

You should get the error: "Violation of PRIMARY KEY constraint Cannot insert duplicate key in object 'dbo.iris\_models'. The duplicate key value is (Naive Bayes)".

That's because the model name was provided by manually typing in "Naive Bayes" as part of the INSERT statement. Assuming you plan to create lots of models, using different parameters or different algorithms on each run, you should consider setting up a metadata scheme so that you can automatically name models and more easily identify them.

- To get around this error, you can make some minor modifications to the SQL wrapper. This example generates a unique model name by appending the current date and time:

```
DECLARE @model varbinary(max);
DECLARE @new_model_name varchar(50)
SET @new_model_name = 'Naive Bayes ' + CAST(GETDATE()as varchar)
SELECT @new_model_name
EXEC generate_iris_model @model OUTPUT;
INSERT INTO iris_models (model_name, model) values(@new_model_name, @model);
```

- To view the models, run a simple SELECT statement.

```
SELECT * FROM iris_models;
GO
```

## Results

MODEL_NAME	MODEL
Naive Bayes	0x800363736B6C656172...
Naive Bayes Jan 01 2018 9:39AM	0x800363736B6C656172...
Naive Bayes Feb 01 2018 10:51AM	0x800363736B6C656172...

## Generate scores from the model

Finally, let's load this model from the table into a variable, and pass it back to Python to generate scores.

- Run the following code to create the stored procedure that performs scoring.

```

CREATE PROCEDURE predict_species (@model varchar(100))
AS
BEGIN
DECLARE @nb_model varbinary(max) = (SELECT model FROM iris_models WHERE model_name = @model);
EXEC sp_execute_external_script @language = N'Python',
@script = N'
import pickle
irismodel = pickle.loads(nb_model)
species_pred = irismodel.predict(iris_data[[1,2,3,4]])
iris_data["PredictedSpecies"] = species_pred
OutputDataSet = iris_data.query(''PredictedSpecies != SpeciesId'')[[0, 5, 6]]
print(OutputDataSet)
'

, @input_data_1 = N'select id, "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width",
"SpeciesId" from iris_data'
, @input_data_1_name = N'iris_data'
, @params = N'@nb_model varbinary(max)'
, @nb_model = @nb_model
WITH RESULT SETS (("id" int, "SpeciesId" int, "SpeciesId.Predicted" int));
END;
GO

```

The stored procedure gets the Naïve Bayes model from the table and uses the functions associated with the model to generate scores. In this example, the stored procedure gets the model from the table using the model name. However, depending on what kind of metadata you are saving with the model, you could also get the most recent model, or the model with the highest accuracy.

- Run the following lines to pass the model name "Naive Bayes" to the stored procedure that executes the scoring code.

```

EXEC predict_species 'Naive Bayes';
GO

```

When you run the stored procedure, it returns a Python data.frame. This line of T-SQL specifies the schema for the returned results: `WITH RESULT SETS ( ("id" int, "SpeciesId" int, "SpeciesId.Predicted" int));`

You can insert the results into a new table, or return them to an application.

This example has been made simple by using the data from the Python iris dataset for scoring. (See the line `iris_data[[1,2,3,4]]`.) However, more typically you would run a SQL query to get the new data, and pass that into Python as `InputDataSet`.

## Remarks

If you are used to working in Python, you might be accustomed to loading data, creating some summaries and graphs, then training a model and generating some scores all in the same 250 lines of code.

However, if your goal is to operationalize the process (model creation, scoring, etc.) in SQL Server, it is important to consider ways that you can separate the process into repeatable steps that can be modified using parameters. As much as possible, you want the Python code that you run in a stored procedure to have clearly defined inputs and outputs that map to stored procedure inputs and outputs.

Moreover, you can generally improve performance by separating the data exploration process from the processes of training a model or generating scores.

Scoring and training processes can often be optimized by leveraging features of SQL Server, such as parallel processing, or by using algorithms in [revoscalepy](#) or [MicrosoftML](#) that support streaming and parallel execution, rather than using standard Python libraries.

## Next lesson

In the final lesson, you run Python code from a remote client, using SQL Server as the compute context. This step is optional, if you don't have a Python client, or don't intend to run Python outside a stored procedure.

- [Create a revoscalepy model from a Python client](#)

# Use Python with revoscalepy to create a model

6/4/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In this lesson, you learn how to run Python code from a remote development client, to create a linear regression model in SQL Server.

## Prerequisites

- This lesson uses different data than the previous lessons. You do not need to complete the previous lessons first. However, if you have completed the previous lessons and have a server already configured to run Python, use that server and database as a compute context.
- To run Python code using SQL Server as a compute context requires SQL Server 2017 or later. Moreover, you must explicitly install and then enable the feature, **Machine Learning Services**, choosing the Python language option.

If you installed a pre-release version of SQL Server 2017, you should update to at least the RTM version. Later service releases have continued to expand and improve Python functionality. Some features of this tutorial might not work in early pre-release versions.

- This example uses a predefined Python environment, named `PYTEST_SQL_SERVER`. The environment has been configured to contain **revoscalepy** and other required libraries.

If you do not have an environment configured to run Python, you must do so separately. A discussion of how to create or modify Python environments is out of scope for this tutorial. For more information about how to set up a Python client that contains the correct libraries, see [Install Python client](#) and [Link Python to tools](#).

## Remote compute contexts and revoscalepy

This sample demonstrates the process of creating a Python model in a remote *compute context*, which lets you work from a client, but choose a remote environment, such as SQL Server, Spark, or Machine Learning Server, where the operations are actually performed. Using compute contexts makes it easier to write code once and deploy it to any supported environment.

To execute Python code in SQL Server requires the **revoscalepy** package. This is a special Python package provided by Microsoft, similar to the **RevoScaleR** package for the R language. The **revoscalepy** package supports the creation of compute contexts, and provides the infrastructure for passing data and models between a local workstation and a remote server. The **revoscalepy** function that supports in-database code execution is `RxInSqlServer`.

In this lesson, you use data in SQL Server to train a linear model based on `rx_lin_mod`, a function in **revoscalepy** that supports regression over very large datasets.

This lesson also demonstrates the basics of how to set up and then use a **SQL Server compute context** in Python. For a discussion of how compute contexts work with other platforms, and which compute contexts are supported, see [Compute context for script execution in Machine Learning Server](#)

## Prepare the database and sample data

1. This lesson uses the database `sqlpy`. If you have not completed any of the previous lessons, you can create the database by running the following code:

```
CREATE DATABASE sqlpy;
GO
USE sqlpy;
GO
```

#### IMPORTANT

If you want to use a different database, be sure to edit the sample code and change the database name in the connection string.

2. This sample uses the Airline dataset, which is available in both R and Python. After you have created a database for your Python samples, populate a table with the data as described here: [Sample data in RevoScaleR](#).
3. Change the name of this environment to use an environment available on your client.

## Run the sample code

After you have prepared the database and have the data for training stored in a table, open a Python development environment and run the code sample.

The code performs the following steps:

1. Imports the required libraries and functions.
2. Creates a connection to SQL Server. Creates **data source** objects for working with the data.
3. Modifies the data using **transformations** so that it can be used by the logistic regression algorithm.
4. Calls `rx_lin_mod` and defines the formula used to fit the model.
5. Generates a set of predictions based on the original data.
6. Creates a summary based on the predicted values.

All operations are performed using an instance of SQL Server as the compute context.

#### NOTE

For a demonstration of this sample running from the command line, see this video: [SQL Server 2017 Advanced Analytics with Python](#)

## Sample code

```

from revoscalepy import RxComputeContext, RxInSqlServer, RxSqlServerData
from revoscalepy import rx_lin_mod, rx_predict, rx_summary
from revoscalepy import RxOptions, rx_import

import os

def test_linmod_sql():
 sql_server = os.getenv('PYTEST_SQL_SERVER', '.')

 sql_connection_string = 'Driver=SQL Server;Server=' + sqlServer +
 ';Database=sqlpy;Trusted_Connection=True;'
 print("connectionString={0!s}".format(sql_connection_string))

 data_source = RxSqlServerData(
 sql_query = "select top 10 * from airlinedemosmall",
 connection_string = sql_connection_string,

 column_info = {
 "ArrDelay" : { "type" : "integer" },
 "DayOfWeek" : {
 "type" : "factor",
 "levels" : ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
 }
 })
)

 sql_compute_context = RxInSqlServer(
 connection_string = sql_connection_string,
 num_tasks = 4,
 auto_cleanup = False
)

 #
 # Run linmod locally
 #
 linmod_local = rx_lin_mod("ArrDelay ~ DayOfWeek", data = data_source)
 #
 # Run linmod remotely
 #
 linmod = rx_lin_mod("ArrDelay ~ DayOfWeek", data = data_source, compute_context = sql_compute_context)

 # Predict results
 #
 predict = rx_predict(linmod, data = rx_import(input_data = data_source))
 summary = rx_summary("ArrDelay ~ DayOfWeek", data = data_source, compute_context = sql_compute_context)

```

## Defining a data source vs. defining a compute context

A data source is different from a compute context. The *data source* defines the data used in your code. The *compute context* defines where the code will be executed. However, they use some of the same information:

- Python variables, such as `sql_query` and `sql_connection_string`, define the source of the data.

Pass these variables to the `RxSqlServerData` constructor to implement the **data source object** named `data_source`.

- You create a **compute context object** by using the `RxInSqlServer` constructor. The resulting **compute context object** is named `sql_cc`.

This example re-uses the same connection string that you used in the data source, on the assumption that the data is on the same SQL Server instance that you will be using as the compute context.

However, the data source and the compute context could be on different servers.

## Changing compute contexts

After you define a compute context, you must set the **active compute context**.

By default, most operations are run locally, which means that if you don't specify a different compute context, the data will be fetched from the data source, and the code will run in your current Python environment.

There are two ways to set the active compute context:

- As an argument of a method or function
- By calling `rx_set_computecontext`

#### Set compute context as an argument of a method or function

In this example, you set the compute context by using an argument of the individual `rx` function.

```
linmod = rx_lin_mod_ex("ArrDelay ~ DayOfWeek", data = data, compute_context = sql_compute_context)
```

This compute context is reused in the call to `rxsummary`:

```
summary = rx_summary("ArrDelay ~ DayOfWeek", data = data_source, compute_context = sql_compute_context)
```

#### Set a compute context explicitly using `rx_set_computecontext`

The function `rx_set_computecontext` lets you toggle between compute contexts that have already been defined.

After you have set the active compute context, it remains active until you change it.

### Using parallel processing and streaming

When you define the compute context, you can also set parameters that control how the data is handled by the compute context. These parameters differ depending on the data source type.

For SQL Server compute contexts, you can set the batch size, or provide hints about the degree of parallelism to use in running tasks.

- The sample was run on a computer with four processors, so the `num_tasks` parameter is set to 4 to allow maximum use of resources.
- If you set this value to 0, SQL Server uses the default, which is to run as many tasks in parallel as possible, under the current MAXDOP settings for the server. However, the exact number of tasks that might be allocated depends on many other factors, such as server settings, and other jobs that are running.

## Related samples

These additional Python samples and tutorials demonstrate end-to-end scenarios using more complex data sources, as well as the use of remote compute contexts.

- [In-Database Python for SQL developers](#)
- [Build a predictive model using Python and SQL Server](#)
- [Build a predictive model using Python and SQL Server](#)

# In-Database Python analytics for SQL developers

4/11/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The goal of this walkthrough is to provide SQL programmers with hands-on experience building a machine learning solution using Python that runs in SQL Server. In this walkthrough, you'll learn how to add Python code to stored procedures and run stored procedures to build and predict from models.

## NOTE

Prefer R? See [this tutorial](#), which provides a similar solution but uses R, and can be run in either SQL Server 2016 or SQL Server 2017.

## Overview

The process of building a machine learning solution is a complex one that can involve multiple tools, and the coordination of subject matter experts across several phases:

- obtaining and cleaning data
- exploring the data and building features useful for modeling
- training and tuning the model
- deployment to production

**The focus of this walkthrough is on building and deploying a solution using SQL Server.**

The data is from the well-known NYC Taxi data set. To make this walkthrough quick and easy, the data is sampled. You'll create a binary classification model that predicts whether a particular trip is likely to get a tip or not, based on columns such as the time of day, distance, and pick-up location.

All tasks can be done using Transact-SQL stored procedures in the familiar environment of Management Studio

- [Step 1: Download the sample data](#)

Download the sample dataset and all script files to a local computer.

- [Step 2: Import data to SQL Server using PowerShell](#)

Execute a PowerShell script that creates a database and a table on the specified instance, and loads the sample data to the table.

- [Step 3: Explore and visualize the data using Python](#)

Perform basic data exploration and visualization, by calling Python from Transact-SQL stored procedures.

- [Step 4: Create data features using Python in T-SQL](#)

Create new data features using custom SQL functions.

- [Step 5: Train and save a Python model using T-SQL](#)

Build and save the machine learning model, using Python in stored procedures.

This walkthrough demonstrates how to perform a binary classification task; you could also use the data

to build models for regression or multiclass classification.

- [Step 6: Operationalize the Python model](#)

After the model has been saved to the database, call the model for prediction using Transact-SQL.

## Requirements

### Prerequisites

- Install an instance of SQL Server 2017 with Machine Learning Services and Python enabled. For more information, see [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#).
- The login that you use for this walkthrough must have permissions to create databases and other objects, to upload data, select data, and run stored procedures.

### Experience level

You should be familiar with fundamental database operations, such as creating databases and tables, importing data into tables, and creating SQL queries.

An experienced SQL programmer should be able to complete this walkthrough by using Transact-SQL in SQL Server Management Studio or by running the provided PowerShell scripts.

Python: Basic knowledge is useful but not required. All Python code is provided.

Some knowledge of PowerShell is helpful.

### Tools

For this tutorial, we're assuming that you've reached the deployment phase. You have been given clean data, complete T-SQL code for feature engineering, and working Python code. Therefore, you can complete this tutorial using SQL Server Management Studio, or any other tool that supports running SQL statements.

- [Overview of SQL Server tools](#)

If you want to develop and test your own Python code, or debug a Python solution, we recommend using a dedicated development environment:

- **Visual Studio 2017** supports both R and Python. We recommend the [Data Science workload](#), which also supports R and F#.
- If you have an earlier version of Visual Studio, [Python Extensions for Visual Studio](#) makes it easy to manage multiple Python environments.
- PyCharm is a popular IDE among Python developers.

**NOTE**

In general, avoid writing or testing new Python code in SQL Server Management Studio. If the code that you embed in a stored procedure has any problems, the information that is returned from the stored procedure is usually inadequate to understand the cause of the error.

Use the following resources to help you plan and execute a successful machine learning project:

- [Team Data Science Process](#)

### Estimated time required

STEP	TIME (HRS)
Download the sample data	0:15

STEP	TIME (HRS)
Import data to SQL Server using PowerShell	0:15
Explore and visualize the data	0:20
Create data features using T-SQL	0:30
Train and save a Model using T-SQL	0:15
Operationalize the model	0:40

## Get started

[Step 1: Download the sample data](#)

# Step 1: Download the sample data

6/8/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

Both the data and the scripts for this tutorial are shared on Github. In this step, you use a PowerShell script to download the data and script files to a local directory of your choosing.

## Run the script

1. Open a Windows PowerShell command console.

Use the option, **Run as Administrator**, if administrative privileges are needed to create the destination directory or to write files to the specified destination.

2. Run the following PowerShell commands, changing the value of the parameter *DestDir* to any local directory. The default we've used here is `C:\temp\pysql`.

```
$source = 'https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/PythonSQL/Download_Scripts_SQL_Walkthrough.ps1'
$ps1_dest = "$pwd\Download_Scripts_SQL_Walkthrough.ps1"
$wc = New-Object System.Net.WebClient
$wc.DownloadFile($source, $ps1_dest)
.\\Download_Scripts_SQL_Walkthrough.ps1 -DestDir 'C:\temp\pysql'
```

If the folder you specify in *DestDir* does not exist, it will be created by the PowerShell script.

If you get an error, temporarily set the policy for execution of PowerShell scripts to **unrestricted** for this walkthrough, by using the **Bypass** argument and scoping the changes to the current session. Running this command does not result in a configuration change.

```
Set-ExecutionPolicy Bypass -Scope Process
```

3. Depending on your internet connection, the download might take a while.

## View results

When all files have been downloaded, the PowerShell script opens to the folder specified by *DestDir*.

- In the PowerShell command prompt, run the following command, to list the files that were downloaded.

```
ls
```

Mode	LastWriteTime	Length	Name
-a----	10/13/2017 1:41 PM	1735	create-db-tb-upload-data.sql
-a----	10/13/2017 1:41 PM	410	DeserializeSavePlots.py
-a----	10/13/2017 1:41 PM	1038	fnCalculateDistance.sql
-a----	10/13/2017 1:41 PM	954	fnEngineerFeatures.sql
-a----	10/13/2017 1:41 PM	351355322	nyctaxi1pct.csv
-a----	10/13/2017 1:41 PM	1346	PredictTipRxPy.sql
-a----	10/13/2017 1:41 PM	1241	PredictTipSciKitPy.sql
-a----	10/13/2017 1:41 PM	1979	PredictTipSingleModeRxPy.sql
-a----	10/13/2017 1:41 PM	1941	PredictTipSingleModeSciKitPy.sql
-a----	10/13/2017 1:41 PM	10762	RunSQL_SQL_Walkthrough.ps1
-a----	10/13/2017 1:41 PM	1561	SerializePlots.sql
-a----	10/13/2017 1:41 PM	3701	taxiimportfmt.xml
-a----	10/13/2017 1:41 PM	721	TrainingTestingSplit.sql
-a----	10/13/2017 1:41 PM	1222	TrainTipPredictionModeIRxPy.sql
-a----	10/13/2017 1:41 PM	1376	TrainTipPredictionModeISciKitPy.sql
-a----	10/13/2017 1:41 PM	78	varbinplot.fmt

## Next step

[Step 2: Import data to SQL Server using PowerShell](#)

## Previous Step

[In-Database Python Analytics for the SQL Developer](#)

## See Also

[Machine Learning Services with Python](#)

# Step 2: Import data to SQL Server using PowerShell

6/8/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

In this step, you run one of the downloaded scripts to create the database objects required for the walkthrough. The script also creates several stored procedures, and uploads the sample data to a table in the database you specified.

## Create database objects and load data

Among the downloaded files you should see a PowerShell script, `RunSQL_SQL_Walkthrough.ps1`. The purpose of this script is to prepare the environment for the walkthrough.

The script performs these actions:

- Installs the SQL Native Client and SQL command-line utilities, if not already installed. These utilities are required for bulk-loading the data to the database using **bcp**.
- Creates a database and a table on the SQL Server instance, and bulk-inserts data into the table.
- Creates multiple SQL functions and stored procedures.

If you run into problems, you can use the script as a reference to perform the steps manually.

### Modify the script to use a trusted Windows identity

By default, the script assumes a SQL Server database user login and password. If you are db\_owner under your Windows user account, you can use your Windows identity to create the objects. To do so, open

`RunSQL_SQL_Walkthrough.ps1` in a code editor to append `-T` to the bcp bulk insert command:

```
bcp $db_tb in $csvfilepath -t ',' -S $server -f taxiimportfmt.xml -F 2 -C "RAW" -b 200000 -U $u -P $p -T
```

### Run the script

1. Open a PowerShell command prompt as administrator. If you are not already in the folder created in the previous step, navigate to the folder, and then run the following command:

```
.\RunSQL_SQL_Walkthrough.ps1
```

2. The script prompts for the following information:

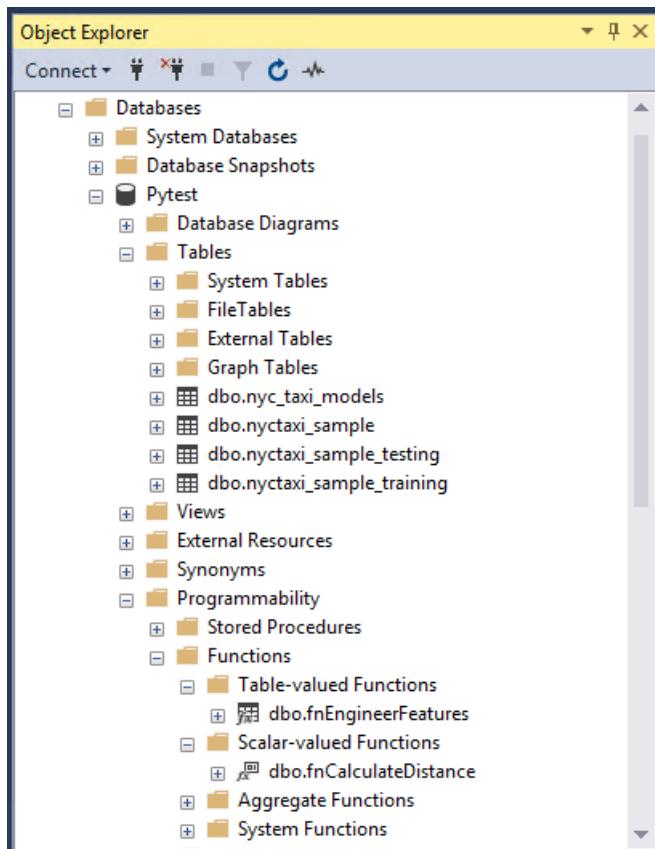
- The name or address of a SQL Server 2017 instance where Machine Learning Services with Python has been installed.
- The user name and password for an account on the instance. The account you use must have the ability to create databases, create tables and stored procedures, and bulk load data to tables.
- If you do not provide the user name and password, your Windows identity is used to sign in to SQL Server.
- The path and file name of the sample data file that you just downloaded. For example,

```
C:\temp\pysql\nyctaxi1pct.csv
```

**NOTE**

To load the data successfully, the library `xmlrw.dll` must be in the same folder as `bcp.exe`.

3. The script also modifies the Transact-SQL scripts that you downloaded earlier, and replaces placeholders with the database name and user name that you provide.
4. When the script completes, log in to the SQL Server instance using the login you specified, to verify that the database, tables, functions, and stored procedures have been created. The following image shows the objects in SQL Server Management Studio.

**NOTE**

If the database objects already existed, they cannot be created again.

If an existing table of the same name and schema is found, the data will be appended, not overwritten. Therefore, be sure to drop or truncate existing tables before running the script.

## List of stored procedures and functions

The following SQL Server objects are created by the script:

SQL SCRIPT FILE NAME	FUNCTION

SQL SCRIPT FILE NAME	FUNCTION
create-db-tb-upload-data.sql	<p>Creates a database and two tables:</p> <p>nyctaxi_sample: Contains the main NYC Taxi dataset. A clustered columnstore index is added to the table to improve storage and query performance. The 1% sample of the NYC Taxi dataset will be inserted into this table.</p> <p>nyc_taxi_models: Used to persist the trained advanced analytics model.</p>
fnCalculateDistance.sql	Creates a scalar-valued function that calculates the direct distance between pickup and dropoff locations
fnEngineerFeatures.sql	Creates a table-valued function that creates new data features for model training
TrainingTestingSplit.sql	Split the data in the nyctaxi_sample table into two parts: nyctaxi_sample_training and nyctaxi_sample_testing.
PredictTipSciKitPy.sql	Creates a stored procedure that calls the trained model (scikit-learn) to create predictions using the model. The stored procedure accepts a query as its input parameter and returns a column of numeric values containing the scores for the input rows.
PredictTipRxPy.sql	Creates a stored procedure that calls the trained model (revoscalepy) to create predictions using the model. The stored procedure accepts a query as its input parameter and returns a column of numeric values containing the scores for the input rows.
PredictTipSingleModeSciKitPy.sql	Creates a stored procedure that calls the trained model (scikit-learn) to create predictions using the model. This stored procedure accepts a new observation as input, with individual feature values passed as in-line parameters, and returns a value that predicts the outcome for the new observation.
PredictTipSingleModeRxPy.sql	Creates a stored procedure that calls the trained model (revoscalepy) to create predictions using the model. This stored procedure accepts a new observation as input, with individual feature values passed as in-line parameters, and returns a value that predicts the outcome for the new observation.

In the latter part of this walkthrough, you create these additional stored procedures:

SQL SCRIPT FILE NAME	FUNCTION
SerializePlots.sql	Creates a stored procedure for data exploration. This stored procedure creates a graphic using Python and then serialize the graph objects.

SQL SCRIPT FILE NAME	FUNCTION
TrainTipPredictionModelSciKitPy.sql	Creates a stored procedure that trains a logistic regression model (scikit-learn). The model predicts the value of the tipped column, and is trained using a randomly selected 60% of the data. The output of the stored procedure is the trained model, which is saved in the table nyc_taxi_models.
TrainTipPredictionModelRxPy.sql	Creates a stored procedure that trains a logistic regression model (revoScaleR). The model predicts the value of the tipped column, and is trained using a randomly selected 60% of the data. The output of the stored procedure is the trained model, which is saved in the table nyc_taxi_models.

## Next step

[Step 3: Explore and visualize the data](#)

## Previous step

[Step 1: Download the sample data](#)

# Step 3: Explore and visualize the data

4/11/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

In this step, you explore the sample data and generate some plots. Later, you learn how to serialize graphics objects in Python, and then deserialize those objects and make plots.

## Review the data

First, take a minute to browse the data schema, as we've made some changes to make it easier to use the NYC Taxi data

- The original dataset used separate files for the taxi identifiers and trip records. We've joined the two original datasets on the columns *medallion*, *hack\_license*, and *pickup\_datetime*.
- The original dataset spanned many files and was quite large. We've downsampled to get just 1% of the original number of records. The current data table has 1,703,957 rows and 23 columns.

### Taxi identifiers

The *medallion* column represents the taxi's unique ID number.

The *hack\_license* column contains the taxi driver's license number (anonymized).

### Trip and fare records

Each trip record includes the pickup and drop-off location and time, and the trip distance.

Each fare record includes payment information such as the payment type, total amount of payment, and the tip amount.

The last three columns can be used for various machine learning tasks. The *tip\_amount* column contains continuous numeric values and can be used as the **label** column for regression analysis. The *tipped* column has only yes/no values and is used for binary classification. The *tip\_class* column has multiple **class labels** and therefore can be used as the label for multi-class classification tasks.

The values used for the label columns are all based on the `tip_amount` column, using these business rules:

- Label column `tipped` has possible values 0 and 1
  - If `tip_amount > 0`, `tipped = 1`; otherwise `tipped = 0`
- Label column `tip_class` has possible class values 0-4

Class 0: `tip_amount = $0`

Class 1: `tip_amount > $0 and tip_amount <= $5`

Class 2: `tip_amount > $5 and tip_amount <= $10`

Class 3: `tip_amount > $10 and tip_amount <= $20`

Class 4: `tip_amount > $20`

# Create plots using Python in T-SQL

Developing a data science solution usually includes intensive data exploration and data visualization. Because visualization is such a powerful tool for understanding the distribution of the data and outliers, Python provides many packages for visualizing data. The **matplotlib** module is one of the more popular libraries for visualization, and includes many functions for creating histograms, scatter plots, box plots, and other data exploration graphs.

In this section, you learn how to work with plots using stored procedures. Rather than open the image on the server, you store the Python object `plot` as **varbinary** data, and then write that to a file that can be shared or viewed elsewhere.

## Create a plot as varbinary data

The **revoscalepy** module included with SQL Server 2017 Machine Learning Services supports features similar to those in the **RevoScaleR** package for R. This example uses the Python equivalent of `rxHistogram` to plot a histogram based on data from a Transact-SQL query.

The stored procedure returns a serialized Python `figure` object as a stream of **varbinary** data. You cannot view the binary data directly, but you can use Python code on the client to deserialize and view the figures, and then save the image file on a client computer.

1. Create the stored procedure *SerializePlots*, if the PowerShell script did not already do so.

- The variable `@query` defines the query text `SELECT tipped FROM nytaxi_sample`, which is passed to the Python code block as the argument to the script input variable, `@input_data_1`.
- The Python script is fairly simple: **matplotlib** `figure` objects are used to make the histogram and scatter plot, and these objects are then serialized using the `pickle` library.
- The Python graphics object is serialized to a **pandas** DataFrame for output.

```

CREATE PROCEDURE [dbo].[SerializePlots]
AS
BEGIN
SET NOCOUNT ON;
DECLARE @query nvarchar(max) =
N'SELECT cast(tipped as int) as tipped, tip_amount, fare_amount FROM [dbo].[nyctaxi_sample]'
```

EXECUTE sp\_execute\_external\_script  
@language = N'Python',  
@script = N'  
import matplotlib  
matplotlib.use("Agg")  
import matplotlib.pyplot as plt  
import pandas as pd  
import pickle

```

fig_handle = plt.figure()
plt.hist(InputDataSet.tipped)
plt.xlabel("Tipped")
plt.ylabel("Counts")
plt.title("Histogram, Tipped")
plot0 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =["plot"])
plt.clf()

plt.hist(InputDataSet.tip_amount)
plt.xlabel("Tip amount ($)")
plt.ylabel("Counts")
plt.title("Histogram, Tip amount")
plot1 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =["plot"])
plt.clf()

plt.hist(InputDataSet.fare_amount)
plt.xlabel("Fare amount ($)")
plt.ylabel("Counts")
plt.title("Histogram, Fare amount")
plot2 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =["plot"])
plt.clf()

plt.scatter(InputDataSet.fare_amount, InputDataSet.tip_amount)
plt.xlabel("Fare Amount ($)")
plt.ylabel("Tip Amount ($)")
plt.title("Tip amount by Fare amount")
plot3 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =["plot"])
plt.clf()

OutputDataSet = plot0.append(plot1, ignore_index=True).append(plot2,
ignore_index=True).append(plot3, ignore_index=True)
',
@input_data_1 = @query
WITH RESULT SETS ((plot varbinary(max)))
END
GO

```

- Now run the stored procedure with no arguments to generate a plot from the data hard-coded as the input query.

```
EXEC [dbo].[SerializePlots]
```

- The results should be something like this:

```
plot
0xFFD8FFE000104A4649...
0xFFD8FFE000104A4649...
0xFFD8FFE000104A4649...
0xFFD8FFE000104A4649...
```

4. From a Python client, you can now connect to the SQL Server instance that generated the binary plot objects, and view the plots.

To do this, run the following Python code, replacing the server name, database name, and credentials as appropriate. Make sure the Python version is the same on the client and the server. Also make sure that the Python libraries on your client (such as matplotlib) are the same or higher version relative to the libraries installed on the server.

#### **Using SQL Server authentication:**

```
import pyodbc
import pickle
import os
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER={SERVER_NAME};DATABASE={DB_NAME};UID={USER_NAME};PWD={PASSWORD}')
cursor = cnxn.cursor()
cursor.execute("EXECUTE [dbo].[SerializePlots]")
tables = cursor.fetchall()
for i in range(0, len(tables)):
 fig = pickle.loads(tables[i][0])
 fig.savefig(str(i)+'.png')
print("The plots are saved in directory: ",os.getcwd())
```

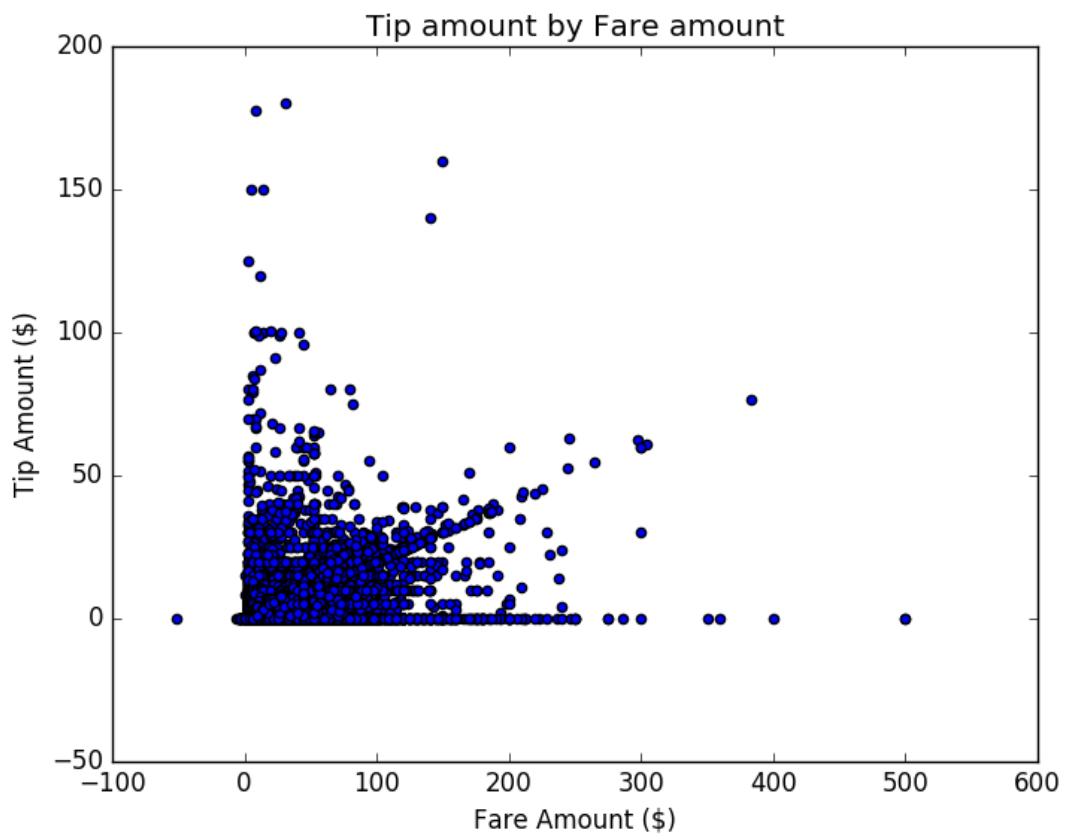
#### **Using Windows authentication:**

```
import pyodbc
import pickle
import os
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER={SERVER_NAME};DATABASE={DB_NAME};Trusted_Connection=yes;')
cursor = cnxn.cursor()
cursor.execute("EXECUTE [dbo].[SerializePlots]")
tables = cursor.fetchall()
for i in range(0, len(tables)):
 fig = pickle.loads(tables[i][0])
 fig.savefig(str(i)+'.png')
print("The plots are saved in directory: ",os.getcwd())
```

5. If the connection is successful, you should see a message like the following:

*The plots are saved in directory: xxxx*

6. The output file is created in the Python working directory. To view the plot, locate the Python working directory, and open the file. The following image shows a plot saved on the client computer.



Next step

[Step 4: Create data features using T-SQL](#)

Previous step

[Step 2: Import data to SQL Server using PowerShell](#)

# Step 4: Create Data Features using T-SQL

4/11/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

After data exploration, you have collected some insights from the data, and are ready to move on to *feature engineering*. This process of creating features from the raw data can be a critical step in advanced analytics modeling.

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

In this step, you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

## Define the Function

The distance values reported in the original data are based on the reported meter distance, and don't necessarily represent geographical distance or distance traveled. Therefore, you'll need to calculate the direct distance between the pick-up and drop-off points, by using the coordinates available in the source NYC Taxi dataset. You can do this by using the [Haversine formula](#) in a custom Transact-SQL function.

You'll use one custom T-SQL function, *fnCalculateDistance*, to compute the distance using the Haversine formula, and use a second custom T-SQL function, *fnEngineerFeatures*, to create a table containing all the features.

### Calculate trip distance using *fnCalculateDistance*

1. The function *fnCalculateDistance* should have been downloaded and registered with SQL Server as part of the preparation for this walkthrough. Take a minute to review the code.

In Management Studio, expand **Programmability**, expand **Functions** and then **Scalar-valued functions**. Right-click *fnCalculateDistance*, and select **Modify** to open the Transact-SQL script in a new query window.

```
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
-- User-defined function that calculates the direct distance between two geographical coordinates
RETURNS float
AS
BEGIN
 DECLARE @distance decimal(28, 10)
 -- Convert to radians
 SET @Lat1 = @Lat1 / 57.2958
 SET @Long1 = @Long1 / 57.2958
 SET @Lat2 = @Lat2 / 57.2958
 SET @Long2 = @Long2 / 57.2958
 -- Calculate distance
 SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2) * COS(@Long2 - @Long1))
 --Convert to miles
 IF @distance <> 0
 BEGIN
 SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) / @distance);
 END
 RETURN @distance
END
GO
```

### Notes:

- The function is a scalar-valued function, returning a single data value of a predefined type.
- It takes latitude and longitude values as inputs, obtained from trip pick-up and drop-off locations. The Haversine formula converts locations to radians and uses those values to compute the direct distance in miles between those two locations.

To add the computed value to a table that can be used for training the model, you'll use another function, *fnEngineerFeatures*.

### Save the features using *fnEngineerFeatures*

1. Take a minute to review the code for the custom T-SQL function, *fnEngineerFeatures*, which should have been created for you as part of the preparation for this walkthrough.

This function is a table-valued function that takes multiple columns as inputs, and outputs a table with multiple feature columns. The purpose of this function is to create a feature set for use in building a model.

The function *fnEngineerFeatures* calls the previously created T-SQL function, *fnCalculateDistance*, to get the direct distance between pickup and dropoff locations.

```
CREATE FUNCTION [dbo].[fnEngineerFeatures] (
 @passenger_count int = 0,
 @trip_distance float = 0,
 @trip_time_in_secs int = 0,
 @pickup_latitude float = 0,
 @pickup_longitude float = 0,
 @dropoff_latitude float = 0,
 @dropoff_longitude float = 0)
RETURNS TABLE
AS
RETURN
(
-- Add the SELECT statement with parameter references here
SELECT
 @passenger_count AS passenger_count,
 @trip_distance AS trip_distance,
 @trip_time_in_secs AS trip_time_in_secs,
 [dbo].[fnCalculateDistance](@pickup_latitude, @pickup_longitude, @dropoff_latitude,
 @dropoff_longitude) AS direct_distance
)
GO
```

2. To verify that this function works, you can use it to calculate the geographical distance for those trips where the metered distance was 0 but the pick-up and drop-off locations were different.

```
SELECT tipped, fare_amount, passenger_count,(trip_time_in_secs/60) as TripMinutes,
trip_distance, pickup_datetime, dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS
direct_distance
FROM nytaxi_sample
WHERE pickup_longitude != dropoff_longitude and pickup_latitude != dropoff_latitude and
trip_distance = 0
ORDER BY trip_time_in_secs DESC
```

As you can see, the distance reported by the meter doesn't always correspond to geographical distance. This is why feature engineering is important.

In the next step, you'll learn how to use these data features to create and train a machine learning model using Python.

## Next step

[Step 5: Train and save a Python model using T-SQL](#)

[Previous step](#)

[Step 3: Explore and visualize the data](#)

# Step 5: Train and save a Python model using T-SQL

4/11/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

In this step, you learn how to train a machine learning model using the Python packages **scikit-learn** and **revoscalepy**. These Python libraries are already installed with SQL Server Machine Learning Services.

You load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure. The model requires the data features you engineered in earlier lessons. Finally, you save the trained model to a SQL Server table.

## IMPORTANT

There have been several changes in the **revoscalepy** package, which required small changes in the code for this tutorial. See the [changelog](#) at the end of this tutorial.

If you installed Python Services using a prerelease version of SQL Server 2017, we recommend that you upgrade to the latest version.

## Split the sample data into training and testing sets

1. You can use the stored procedure **TrainTestSplit** to divide the data in the nyctaxi\_sample table into two parts: nyctaxi\_sample\_training and nyctaxi\_sample\_testing.

This stored procedure should already be created for you, but you can run the following code to create it:

```
CREATE PROCEDURE [dbo].[TrainTestSplit] (@pct int)
AS

DROP TABLE IF EXISTS dbo.nyctaxi_sample_training
SELECT * into nyctaxi_sample_training FROM nyctaxi_sample WHERE
(ABS(CAST(BINARY_CHECKSUM(medallion,hack_license) as int)) % 100) < @pct

DROP TABLE IF EXISTS dbo.nyctaxi_sample_testing
SELECT * into nyctaxi_sample_testing FROM nyctaxi_sample
WHERE (ABS(CAST(BINARY_CHECKSUM(medallion,hack_license) as int)) % 100) > @pct
GO
```

2. To divide your data using a custom split, run the stored procedure, and type an integer that represents the percentage of data allocated to the training set. For example, the following statement would allocate 60% of data to the training set.

```
EXEC TrainTestSplit 60
GO
```

## Build a logistic regression model

After the data has been prepared, you can use it to train a model. You do this by calling a stored procedure that runs some Python code, taking as input the training data table. For this tutorial, you create two models, both

binary classification models:

- The stored procedure **TrainTipPredictionModelRxPy** creates a tip prediction model using the **revoscalepy** package.
- The stored procedure **TrainTipPredictionModelSciKitPy** creates a tip prediction model using the **scikit-learn** package.

Each stored procedure uses the input data you provide to create and train a logistic regression model. All Python code is wrapped in the system stored procedure, [sp\\_execute\\_external\\_script](#).

To make it easier to retrain the model on new data, you wrap the call to `sp_execute_external_script` in another stored procedure, and pass in the new training data as a parameter. This section will walk you through that process.

### TrainTipPredictionModelSciKitPy

1. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure `TrainTipPredictionModelSciKitPy`. The stored procedure contains a definition of the input data, so you don't need to provide an input query.

```
DROP PROCEDURE IF EXISTS TrainTipPredictionModelSciKitPy;
GO

CREATE PROCEDURE [dbo].[TrainTipPredictionModelSciKitPy] (@trained_model varbinary(max) OUTPUT)
AS
BEGIN
 EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
import numpy
import pickle
from sklearn.linear_model import LogisticRegression

##Create SciKit-Learn logistic regression model
X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]
y = numpy.ravel(InputDataSet[["tipped"]])

SKLalgo = LogisticRegression()
logitObj = SKLalgo.fit(X, y)

##Serialize model
trained_model = pickle.dumps(logitObj)
',
 @input_data_1 = N'
select tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
direct_distance
from nyctaxi_sample_training
',
 @input_data_1_name = N'InputDataSet',
 @params = N'@trained_model varbinary(max) OUTPUT',
 @trained_model = @trained_model OUTPUT;
;
END;
GO
```

2. Run the following SQL statements to insert the trained model into table `nyc_taxi_models`.

```
DECLARE @model VARBINARY(MAX);
EXEC TrainTipPredictionModelSciKitPy @model OUTPUT;
INSERT INTO nyc_taxi_models (name, model) VALUES('SciKit_model', @model);
```

Processing of the data and fitting the model might take a couple of mins. Messages that would be piped to Python's **stdout** stream are displayed in the **Messages** window of Management Studio. For example:

*STDOUT message(s) from external script: C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON\_SERVICES\lib\site-packages\revoscalepy*

3. Open the table *nyc\_taxi\_models*. You can see that one new row has been added, which contains the serialized model in the column *model*.

*linear\_model 0x800363736B6C6561726E2E6C696E6561....*

### TrainTipPredictionModelRxPy

This stored procedure uses the new **revoscalepy** package, which is a new package for Python. It contains objects, transformation, and algorithms similar to those provided for the R language's **RevoScaleR** package.

By using **revoscalepy**, you can create remote compute contexts, move data between compute contexts, transform data, and train predictive models using popular algorithms such as logistic and linear regression, decision trees, and more. For more information, see [what is revoscalepy?](#)

1. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure *TrainTipPredictionModelRxPy*. Because the stored procedure already includes a definition of the input data, you don't need to provide an input query.

```
DROP PROCEDURE IF EXISTS TrainTipPredictionModelRxPy;
GO

CREATE PROCEDURE [dbo].[TrainTipPredictionModelRxPy] (@trained_model varbinary(max) OUTPUT)
AS
BEGIN
EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
import numpy
import pickle
from revoscalepy.functions.RxLogit import rx_logit

Create a logistic regression model using rx_logit function from revoscalepy package
logitObj = rx_logit("tipped ~ passenger_count + trip_distance + trip_time_in_secs + direct_distance",
data = InputDataSet);

Serialize model
trained_model = pickle.dumps(logitObj)
',
@input_data_1 = N'
select tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
direct_distance
from nytaxi_sample_training
',
@input_data_1_name = N'InputDataSet',
@params = N'@trained_model varbinary(max) OUTPUT',
@trained_model = @trained_model OUTPUT;
';
END;
GO
```

This stored procedure performs the following steps as part of model training:

- The SELECT query applies the custom scalar function *fnCalculateDistance* to calculate the direct distance between the pick-up and drop-off locations. The results of the query are stored in the default Python input variable, `InputDataset`.
- The binary variable *tipped* is used as the *label* or outcome column, and the model is fit using these

feature columns: *passenger\_count*, *trip\_distance*, *trip\_time\_in\_secs*, and *direct\_distance*.

- The trained model is serialized and stored in the Python variable `logitobj`. By adding the T-SQL keyword OUTPUT, you can add the variable as an output of the stored procedure. In the next step, that variable is used to insert the binary code of the model into a database table *nyc\_taxi\_models*. This mechanism makes it easy to store and re-use models.
2. Run the stored procedure as follows to insert the trained **revoscalepy** model into the table *nyc\_taxi\_models*.

```
DECLARE @model VARBINARY(MAX);
EXEC TrainTipPredictionModelRxPy @model OUTPUT;

INSERT INTO nyc_taxi_models (name, model) VALUES('revoscalepy_model', @model);
```

Processing of the data and fitting the model might take a while. Messages that would be piped to Python's **stdout** stream are displayed in the **Messages** window of Management Studio. For example:

*STDOUT message(s) from external script: C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON\_SERVICES\lib\site-packages\revoscalepy*

3. Open the table *nyc\_taxi\_models*. You can see that one new row has been added, which contains the serialized model in the column *model*.

*rx\_model 0x8003637265766F7363616c....*

In the next step, you use the trained models to create predictions.

## Next step

[Step 6: Operationalize the Python model using SQL Server](#)

## Previous step

[Step 4: Create data features using T-SQL](#)

# Step 6: Operationalize the Python model using SQL Server

4/11/2018 • 9 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is part of a tutorial, [In-database Python analytics for SQL developers](#).

In this step, you learn to *operationalize* the models that you trained and saved in the previous step.

In this scenario, operationalization means deploying the model to production for scoring. The integration with SQL Server makes this fairly easy, because you can embed Python code in a stored procedure. To get predictions from the model based on new inputs, just call the stored procedure from an application and pass the new data.

This lesson demonstrates two methods for creating predictions based on a Python model: batch scoring, and scoring row by row.

- **Batch scoring:** To provide multiple rows of input data, pass a SELECT query as an argument to the stored procedure. The result is a table of observations corresponding to the input cases.
- **Individual scoring:** Pass a set of individual parameter values as input. The stored procedure returns a single row or value.

All the Python code needed for scoring is provided as part of the stored procedures.

STORED PROCEDURE NAME	BATCH OR SINGLE	MODEL SOURCE
PredictTipRxPy	batch	revoscalepy model
PredictTipSciKitPy	batch	scikit-learn model
PredictTipSingleModeRxPy	single row	revoscalepy model
PredictTipSingleModeSciKitPy	single row	scikit-learn model

## Batch scoring

The first two stored procedures illustrate the basic syntax for wrapping a Python prediction call in a stored procedure. Both stored procedures require a table of data as inputs.

- The name of the exact model to use is provided as input parameter to the stored procedure. The stored procedure loads the serialized model from the database table `nyc_taxi_models.table`, using the SELECT statement in the stored procedure.
- The serialized model is stored in the Python variable `mod` for further processing using Python.
- The new cases that need to be scored are obtained from the Transact-SQL query specified in `@input_data_1`. As the query data is read, the rows are saved in the default data frame, `InputDataSet`.
- Both stored procedure use functions from `sklearn` to calculate an accuracy metric, AUC (area under curve). Accuracy metrics such as AUC can only be generated if you also provide the target label (the *tipped* column). Predictions do not need the target label (variable `y`), but the accuracy metric calculation does.

Therefore, if you don't have target labels for the data to be scored, you can modify the stored procedure to

remove the AUC calculations, and return only the tip probabilities from the features (variable `x` in the stored procedure).

## PredictTipSciKitPy

The stored procedure should have already been created for you. If you can't find it, run the following T-SQL statements to create the stored procedures.

This stored procedure requires a model based on the scikit-learn package, because it uses functions specific to that package:

- The data frame containing inputs is passed to the `predict_proba` function of the logistic regression model, `mod`. The `predict_proba` function (`probArray = mod.predict_proba(X)`) returns a **float** that represents the probability that a tip (of any amount) will be given.

```
CREATE PROCEDURE [dbo].[PredictTipSciKitPy] (@model varchar(50), @inquery nvarchar(max))
AS
BEGIN
 DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models where name = @model);
 EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
 import pickle;
 import numpy;
 from sklearn import metrics

 mod = pickle.loads(lmodel2)

 X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]
 y = numpy.ravel(InputDataSet[["tipped"]])

 probArray = mod.predict_proba(X)
 probList = []
 for i in range(len(probArray)):
 probList.append((probArray[i])[1])

 probArray = numpy.asarray(probList)
 fpr, tpr, thresholds = metrics.roc_curve(y, probArray)
 aucResult = metrics.auc(fpr, tpr)
 print ("AUC on testing data is: " + str(aucResult))

 OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
 ,

 @input_data_1 = @inquery,
 @input_data_1_name = N'InputDataSet',
 @params = N'@lmodel2 varbinary(max)',
 @lmodel2 = @lmodel2
 WITH RESULT SETS ((Score float));
 END
GO
```

## PredictTipRxPy

This stored procedure uses the same inputs and creates the same type of scores as the previous stored procedure, but uses functions from the **revoscalepy** package provided with SQL Server machine learning.

### NOTE

The code for this stored procedure has changed slightly between early release versions and the RTM version, to reflect changes to the revoscalepy package. See the [Changes](#) table for details.

```

CREATE PROCEDURE [dbo].[PredictTipRxPy] (@model varchar(50), @inquery nvarchar(max))
AS
BEGIN
 DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models2 where name = @model);

 EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
 import pickle;
 import numpy;
 from sklearn import metrics
 from revoscalepy.functions.RxPredict import rx_predict;

 mod = pickle.loads(lmodel2)
 X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]
 y = numpy.ravel(InputDataSet[["tipped"]])

 probArray = rx_predict(mod, X)
 prob_list = prob_array["tipped_Pred"].values

 probArray = numpy.asarray(probList)
 fpr, tpr, thresholds = metrics.roc_curve(y, probArray)
 aucResult = metrics.auc(fpr, tpr)
 print ("AUC on testing data is: " + str(aucResult))
 OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
 ',
 @input_data_1 = @inquery,
 @input_data_1_name = N'InputDataSet',
 @params = N'@lmodel2 varbinary(max)',
 @lmodel2 = @lmodel2
 WITH RESULT SETS ((Score float));
END
GO

```

## Run batch scoring using a SELECT query

The stored procedures **PredictTipSciKitPy** and **PredictTipRxPy** require two input parameters:

- The query that retrieves the data for scoring
- The name of a trained model

By passing those arguments to the stored procedure, you can select a particular model or change the data used for scoring.

1. To use the **scikit-learn** model for scoring, call the stored procedure **PredictTipSciKitPy**, passing the model name and query string as inputs.

```

DECLARE @query_string nvarchar(max) -- Specify input query
SET @query_string='
 select tipped, fare_amount, passenger_count, trip_time_in_secs, trip_distance,
 dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) as
 direct_distance
 from nyctaxi_sample_testing'
EXEC [dbo].[PredictTipSciKitPy] 'SciKit_model', @query_string;

```

The stored procedure returns predicted probabilities for each trip that was passed in as part of the input query.

If you are using SSMS (SQL Server Management Studio) for running queries, the probabilities will appear as a table in the **Results** pane. The **Messages** pane outputs the accuracy metric (AUC or area under curve) with a value of around 0.56.

2. To use the **revoscalepy** model for scoring, call the stored procedure **PredictTipRxPy**, passing the model name and query string as inputs.

```
EXEC [dbo].[PredictTipRxPy] 'revoscalepy_model', @query_string;
```

## Single-row scoring

Sometimes, instead of batch scoring, you might want to pass in a single case, getting values from an application, and returning a single result based on those values. For example, you could set up an Excel worksheet, web application, or report to call the stored procedure and pass to it inputs typed or selected by users.

In this section, you'll learn how to create single predictions by calling two stored procedures:

- [PredictTipSingleModeSciKitPy](#) is designed for single-row scoring using the scikit-learn model.
- [PredictTipSingleModeRxPy](#) is designed for single-row scoring using the revoscalepy model.
- If you haven't trained a model yet, return to [Step 5!](#)

Both models take as input a series of single values, such as passenger count, trip distance, and so forth. A table-valued function, `fnEngineerFeatures`, is used to convert latitude and longitude values from the inputs to a new feature, direct distance. [Lesson 4](#) contains a description of this table-valued function.

Both stored procedures create a score based on the Python model.

### NOTE

It is important that you provide all the input features required by the Python model when you call the stored procedure from an external application. To avoid errors, you might need to cast or convert the input data to a Python data type, in addition to validating data type and data length.

### PredictTipSingleModeSciKitPy

Take a minute to review the code of the stored procedure that performs scoring using the **scikit-learn** model.

```

CREATE PROCEDURE [dbo].[PredictTipSingleModeSciKitPy] (@model varchar(50), @passenger_count int = 0,
 @trip_distance float = 0,
 @trip_time_in_secs int = 0,
 @pickup_latitude float = 0,
 @pickup_longitude float = 0,
 @dropoff_latitude float = 0,
 @dropoff_longitude float = 0)
AS
BEGIN
 DECLARE @inquery nvarchar(max) = N'
 SELECT * FROM [dbo].[fnEngineerFeatures](
 @passenger_count,
 @trip_distance,
 @trip_time_in_secs,
 @pickup_latitude,
 @pickup_longitude,
 @dropoff_latitude,
 @dropoff_longitude
)'

 DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models2 where name = @model);
 EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
 import pickle;
 import numpy;

 # Load model and unserialize
 mod = pickle.loads(model)

 # Get features for scoring from input data
 X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]

 # Score data to get tip prediction probability as a list (of float)
 probList = []
 probList.append((mod.predict_proba(X)[0])[1])

 # Create output data frame
 OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
 ',
 @input_data_1 = @inquery,
 @params = N'@model varbinary(max),@passenger_count int,@trip_distance float,
 @trip_time_in_secs int ,
 @pickup_latitude float ,
 @pickup_longitude float ,
 @dropoff_latitude float ,
 @dropoff_longitude float',
 @model = @lmodel2,
 @passenger_count =@passenger_count ,
 @trip_distance=@trip_distance,
 @trip_time_in_secs=@trip_time_in_secs,
 @pickup_latitude=@pickup_latitude,
 @pickup_longitude=@pickup_longitude,
 @dropoff_latitude=@dropoff_latitude,
 @dropoff_longitude=@dropoff_longitude
 WITH RESULT SETS ((Score float));
END
GO

```

## PredictTipSingleModeRxPy

The following stored procedure performs scoring using the **revoscalepy** model.

```

CREATE PROCEDURE [dbo].[PredictTipSingleModeRxPy] (@model varchar(50), @passenger_count int = 0,
 @trip_distance float = 0,
 @trip_time_in_secs int = 0,
 @pickup_latitude float = 0,
 @pickup_longitude float = 0,
 @dropoff_latitude float = 0,
 @dropoff_longitude float = 0)
AS
BEGIN
DECLARE @inquery nvarchar(max) = N'
 SELECT * FROM [dbo].[fnEngineerFeatures](
 @passenger_count,
 @trip_distance,
 @trip_time_in_secs,
 @pickup_latitude,
 @pickup_longitude,
 @dropoff_latitude,
 @dropoff_longitude)
 '

DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models2 where name = @model);
EXEC sp_execute_external_script
 @language = N'Python',
 @script = N'
 import pickle;
 import numpy;
 from revoscalepy.functions.RxPredict import rx_predict;

 # Load model and unserialize
 mod = pickle.loads(model)

 # Get features for scoring from input data
 X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]

 # Score data to get tip prediction probability as a list (of float)

 probArray = rx_predict(mod, X)

 probList = []
 prob_list = prob_array["tipped_Pred"].values

 # Create output data frame
 OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
 ',

@input_data_1 = @inquery,
@params = N'@model varbinary(max),@passenger_count int,@trip_distance float,
 @trip_time_in_secs int ,
 @pickup_latitude float ,
 @pickup_longitude float ,
 @dropoff_latitude float ,
 @dropoff_longitude float',
 @model = @lmodel2,
 @passenger_count =@passenger_count ,
 @trip_distance=@trip_distance,
 @trip_time_in_secs=@trip_time_in_secs,
 @pickup_latitude=@pickup_latitude,
 @pickup_longitude=@pickup_longitude,
 @dropoff_latitude=@dropoff_latitude,
 @dropoff_longitude=@dropoff_longitude
WITH RESULT SETS ((Score float));
END
GO

```

## Generate scores from models

After the stored procedures have been created, it is easy to generate a score based on either model. Just open a new **Query** window, and type or paste parameters for each of the feature columns. The seven required values are

for these feature columns, in order:

- *passenger\_count*
- *trip\_distance* *vtrip\_time\_in\_secs*
- *pickup\_latitude*
- *pickup\_longitude*
- *dropoff\_latitude*
- *dropoff\_longitude*

1. To generate a prediction by using the **revoscalepy** model, run this statement:

```
EXEC [dbo].[PredictTipSingleModeRxPy] 'revoscalepy_model', 1, 2.5, 631, 40.763958,-73.973373,
40.782139,-73.977303
```

2. To generate a score by using the **scikit-learn** model, run this statement:

```
EXEC [dbo].[PredictTipSingleModeSciKitPy] 'linear_model', 1, 2.5, 631, 40.763958,-73.973373,
40.782139,-73.977303
```

The output from both procedures is a probability of a tip being paid for the taxi trip with the specified parameters or features.

## Changes

This section lists changes to the code used in this tutorial. These changes were made to reflect the latest **revoscalepy** version. For API help, see [Python function library reference](#).

CHANGE DETAILS	NOTES
deleted <code>import pandas</code> in all samples	pandas now loaded by default
function <code>rx_predict_ex</code> changed to <code>rx_predict</code>	RTM and pre-release versions require <code>rx_predict_ex</code>
function <code>rx_logit_ex</code> changed to <code>rx_logit</code>	RTM and pre-release versions require <code>rx_logit_ex</code>
<code>probList.append(probArray._results["tipped_Pred"])</code> changed to <code>prob_list = prob_array["tipped_Pred"].values</code>	updates to API

If you installed Python Services using a prerelease version of SQL Server 2017, we recommend that you upgrade. You can also upgrade just the Python and R components by using the latest release of Machine Learning Server. For more information, see [Using binding to upgrade an instance of SQL Server](#).

## Conclusions

In this tutorial, you've learned how to work with Python code embedded in stored procedures. The integration with Transact-SQL makes it much easier to deploy Python models for prediction and to incorporate model retraining as part of an enterprise data workflow.

## Previous step

[Step 5: Train and save a Python model](#)

## See also

[Machine Learning Services with Python](#)

# Data science scenarios and solution templates

4/11/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Templates are sample solutions that demonstrate best practices and provide building blocks to help you implement a solution fast. Each template is designed to solve a specific problem, for a specific vertical or industry. The tasks in each template extend from data preparation and feature engineering to model training and scoring. Use these templates to learn how R Services (In-Database) works. Then, feel free to customize the template to fit your own scenario and build a custom solution.

Each solution includes sample data, R code or Python code, and SQL stored procedures if applicable. The code can be run in your preferred R or Python development environment, with computations done in SQL Server. In some cases, you can run code directly using T-SQL and any SQL client tool, such as SQL Server Management Studio.

## TIP

Most of the templates come in multiple versions supporting both on-premises and cloud platforms for machine learning. For example, you can build the solution using only SQL Server, or you can build the solution in Microsoft R Server, or in Azure Machine Learning.

- For details and updates, see this announcement: [Exciting new templates in Azure ML](#)
- For download and setup instructions, see [How to use the templates](#).

## Fraud detection

[Online fraud detection template \(SQL Server R Services\)](#)

**What:** The ability to detect fraudulent transactions is important for online businesses. To reduce charge-back losses, businesses need to quickly identify transactions that were made using stolen payment instruments or credentials. When fraudulent transactions are discovered, businesses typically take measures to block certain accounts as soon as possible, to prevent further losses. In this scenario, you learn how to use data from online purchase transactions to identify likely fraud.

**How:** Fraud detection is solved as a binary classification problem. The methodology used in this template can be easily applied to fraud detection in other domains.

## Campaign optimization

[Predict how and when to contact leads](#)

**What:** This solution uses insurance industry data to predict leads based on demographics, historical response data, and product-specific details. Recommendations are also generated to indicate the best channel and time to approach users to influence purchase behavior.

**How:** The solution creates and compares multiple models. The solution also demonstrates automated data integration and data preparation using stored procedures. Parallel samples are provided for SQL Server in-database, in Azure, and HDInsight Spark.

## Health care: predict length of stay in hospital

## Predicting length of stay in hospitals

**What:** Accurately predicting which patients might require long-term hospitalization is an important part of both care and planning. Administrators need to be able to determine which facilities require more resources, and caregivers want to guarantee that they can meet the needs of patients.

**How:** This solution uses the Data Science Virtual Machine, and includes an instance of SQL Server with machine learning enabled. It also includes a set of Power BI reports that you can use to interact with a deployed model.

## Customer churn

### [Customer churn prediction template \(SQL Server R Services\)](#)

**What:** Analyzing and predicting customer churn is important in any industry where the loss of customers to competitors must be managed and prevented: banking, telecommunications, and retail, to name a few. The goal of churn analysis is to identify which customers are likely to churn, and then take appropriate actions to retain such customers and keep their business.

**How:** This template formulates the churn problem as a **binary classification** problem. It uses sample data from two sources, customer demographics and customer transactions, to classify customers as likely or unlikely to churn.

## Predictive maintenance

### [Predictive maintenance template \(SQL Server 2016\)](#)

**What:** Predictive maintenance aims to increase the efficiency of maintenance tasks by capturing past failures and using that information to predict when or where a device might fail. The ability to forecast device obsolescence is especially valuable for applications that rely on distributed data or sensors. This method could also be applied to monitor or predict error in IoT (Internet of Things) devices.

See this announcement for more information: [New predictive maintenance template](#)

**How:** This solution focuses on answering the question, "When will an in-service machine fail?" The input data represents simulated sensor measurements for aircraft engines. Data obtained from monitoring the engine's current operation conditions, such as the current working cycle, settings, and sensor measurements, are used to create three types of predictive models:

- **Regression models**, to predict how much longer an engine will last before it fails. The sample model predicts the metric "Remaining Useful Life" (RUL), also called "Time to Failure" (TTF).
- **Classification models**, to predict whether an engine is likely to fail.

The **binary classification model** predicts if an engine will fail within a certain time frame.

The **multi-class classification model** predicts whether a particular engine might fail, and provides a probable time window of failure. For example, for a given day, you can predict whether any device is likely to fail on the given day, or in some time period following the given day.

## Energy demand forecasting

### [Energy demand forecasting template with SQL Server R Services](#)

**What:** Demand forecasting is an important problem in various domains including energy, retail, and services. Accurate demand forecasting helps companies conduct better production planning, resource allocation, and make other important business decisions. In the energy sector, demand forecasting is critical for reducing energy storage cost and balancing supply and demand.

**How:** This template uses SQL Server R Services to predict demand for electricity. The model used for prediction is a random forest regression model based on **rxDForest**, a high-performance machine learning algorithm included in Microsoft R Server. The solution includes a demand simulator, all the R and T-SQL code needed to train a model, and stored procedures that you can use to generate and report predictions.

## How to use the templates

To download the files included in each template, you can use GitHub commands, or you can open the link and click **Download Zip** to save all files to your computer. When downloaded, the solution typically contains these folders:

- **Data:** Contains the sample data for each application.
- **R:** Contains all the R development code you need for the solution. The solution requires the libraries provided by Microsoft R Server, but can be opened and edited in any R IDE. The R code has been optimized so that computations are performed "in-database", by setting the compute context to a SQL Server instance.
- **SQLR:** Contains multiple .sql files that you can run in a SQL environment such as SQL Server Management Studio to create the stored procedures that perform related tasks such as data processing, feature engineering, and model deployment.

The folder also contains a PowerShell script that you can run to invoke all scripts and create the end-to-end environment. Be sure to edit the script to suit your environment.

## Next steps

- [SQL Server machine learning tutorials](#)

# Machine learning lifecycle and personas

4/12/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Machine learning projects can be complex, because they require the skills and collaboration of a disparate set of professionals. This article describes the principal tasks in the machine learning lifecycle, the type of data professionals who are engaged in machine learning, and how SQL Server supports the needs.

## TIP

Before you get started on a machine learning project, we recommend that you review the tools and best practices provided by the [Microsoft Team Data Science Process](#), or TDSP. This process was created by machine learning consultants at Microsoft to consolidate best practices around planning and iterating on machine learning projects. The TDSP has its roots in industry standards such as CRISP-DM, but incorporates recent practices such as DevOps and visualization.

## Machine learning life cycle

Machine learning is a complex process that touches all aspects of data in the enterprise, and many machine learning projects end up taking longer and being more complex than anticipated. Here are some of the ways that machine learning requires the support of data professionals in the enterprise:

- Machine learning begins with identification of goals and business rules.
- Machine learning professionals must be aware of policies for storing, extracting, and auditing data.
- Collection of potentially applicable data is next. Data sources must be identified, and the appropriate data extracted from sensors and business applications.
- The quality of machine learning efforts is highly dependent on not just the type of data that is available, but the very processes used for extracting, processing, and storing data.
- No machine learning project would be complete without a strategy for reporting and analysis, and possibly customer engagement and feedback.

SQL Server helps bridge many of the gaps between enterprise data professionals and machine learning experts:

- Data can be stored on-premises or in the cloud
- SQL Server is integrated at every stage of enterprise data processing, including reporting and ETL
- SQL Server supports data security, data redundancy, and auditing
- Provides resource governance

## Data scientists

Data scientists use a variety of tools for data analysis and machine learning, ranging from Excel or free open-source platforms, to expensive statistical suites that require deep technical knowledge. However, using R or Python with SQL Server provides some unique benefits in comparison to these traditional tools:

- You can develop and test a solution by using the development environment of your choice, then deploy your R or Python code as part of T-SQL code.
- Move complex computations off the data scientist's laptop and onto the server, avoiding data movement to comply with enterprise security policies.
- Performance and scale are improved through special R packages and APIs. You are no longer restricted by the

single-threaded, memory-bound architecture of R, and can work with large datasets and multi-threaded, multi-core, multi-process computations.

- Code portability: Solutions can run in SQL Server, or in Hadoop or Linux, using [Machine Learning Server](#).  
Code once, deploy anywhere.

## Application and database developers

Database developers are tasked with integrating multiple technologies and bringing together the results so that they can be shared throughout the enterprise. The database developer works with application developers, SQL developers, and data scientists to design solutions, recommend data management methods, and architect or deploy solutions.

Integration with SQL Server provides many benefits to data developers:

- The data scientist can work in RStudio, while the data developer deploys the solution using SQL Server Management Studio. No more recoding of R or Python solutions.
- Optimize your solutions by using the best of T-SQL, R, and Python. Complex operations on large datasets can be run far more efficiently using SQL Server than in R. Leverage the knowledge of your database professionals to improve the performance of machine learning solutions, by using in-memory columnstore indexes, and aggregations using SQL set-based operations.
- Effortlessly automate tasks that must run repeatedly on large amounts of data, such as generating prediction scores on production data.
- Access parameterized R or Python script from any application that uses Transact-SQL. Just call a stored procedure to train a model, generate a plot, or output predictions.
- APIs can stream large datasets and benefit from multi-threaded, multi-core, multi-process in-database computations.

For information on related tasks, see:

- [Operationalizing your R code](#)

### Database administrators

Database administrators must integrate competing projects and priorities into a single point of contact: the database server. They must provide data access not just to data scientists but to a variety of report developers, business analysts, and business data consumers, while maintaining the health of operational and reporting data stores. In the enterprise, the DBA is a critical part of building and deploying an effective infrastructure for data science.

SQL Server provides unique features for the database administrator who must support the data science role:

- Security by SQL Server: The architecture of R Services (In-Database) keeps your databases secure and isolates the execution of external script sessions from the operation of the database instance. You can specify who has permission to execute machine learning scripts, and use database roles to manage packages.
- R and Python sessions are executed in a separate process to ensure that your server continues to run as usual even if the external script encounters issues.
- Resource governance using SQL Server lets you control the memory and processes allocated to external runtimes, to prevent massive computations from jeopardizing the overall server performance.

For information on related tasks, see:

- [Managing and monitoring machine learning solutions](#)

## Architects and data engineers

Architects design integrated workflows that span all aspects of the machine learning life cycle. Data engineers design and build ETL solutions and determine how to optimize feature engineering tasks for machine learning. The overall data platform must be designed to balance competing business needs.

Because R Services (In-Database) is deeply integrated with other Microsoft tools such as the business intelligence and data warehouse stack, enterprise cloud and mobility tools, and Hadoop, it provides an array of benefits to the data engineer or system architect who wants to promote advanced analytics.

- Call any Python or R script by using system stored procedures, to populate datasets, generate graphics, or get predictions. No more designing parallel workflows in data science and ETL tools. Support for Azure Data Factory and Azure SQL Database makes it easier to use cloud data sources in machine learning workflows.
- For scheduling and management of machine learning tasks, use standard automation workflows in SQL Server, based on Integration Services, SQL Agent, or Azure Data Factory. Or, use the [operationalization features](#) in Machine Learning Server.

For information on related tasks, see:

- [Creating machine learning workflows in SQL Server](#)

# Default R and Python packages in SQL Server

6/1/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article lists the R and Python packages installed with SQL Server and where to find the package library.

## R package list for SQL Server

R packages are installed with [SQL Server 2016 R Services](#) and [SQL Server 2017 Machine Learning Services](#) when you select the R feature during setup.

PACKAGES	2016	2017	DESCRIPTION
RevoScaleR	8.0.3	9.2	Used for remote compute contexts, streaming, parallel execution of rx functions for data import and transformation, modeling, visualization, and analysis.
sqlrutils	8.0.3	9.2	Used for including R script in stored procedures.
MicrosoftML	n.a.	9.2	Adds machine learning algorithms in R.
olapR	n.a.	9.2	Used for writing MDX statements in R.

MicrosoftML and olapR are available by default in SQL Server 2017 Machine Learning Services. On a SQL Server 2016 R Services instance, you can add these packages through a [component upgrade](#). A component upgrade also gets you newer versions of packages (for example, newer versions of RevoScaleR include functions for package management on SQL Server).

## Python package list for SQL Server

Python packages are available only in SQL Server 2017 when you install [SQL Server 2017 Machine Learning Services](#) and select the Python feature.

PACKAGES	2017	DESCRIPTION
revoscalepy	9.2	Used for remote compute contexts, streaming, parallel execution of rx functions for data import and transformation, modeling, visualization, and analysis.
microsoftml	9.2	Adds machine learning algorithms in Python.

## Open-source R in your installation

R support includes open-source R so that you can call base R functions and install additional open-source and third-party packages. R language support includes core functionality such as **base**, **stats**, **utils**, and others. A base installation of R also includes numerous sample datasets, and standard R tools such as **RGui** (a lightweight interactive editor) and **RTerm** (an R command prompt).

The distribution of open-source R included in your installation is [Microsoft R Open \(MRO\)](#). MRO adds value to base R by including additional open-source packages such as the [Intel Math Kernel Library](#).

The following table summarizes the versions of R provided by MRO using SQL Server Setup.

RELEASE	R VERSION
SQL Server 2016 R Services	3.2.2
SQL Server 2017 Machine Learning Services	3.3.3

You should never manually overwrite the version of R installed by SQL Server Setup with newer versions on the web. Microsoft R packages are based on specific versions of R. Modifying your installation could destabilize it.

## Open-source Python in your installation

SQL Server 2017 adds Python components. When you select the Python language option, Anaconda 4.2 distribution is installed. In addition to Python code libraries, the standard installation includes sample data, unit tests, and sample scripts.

SQL Server 2017 Machine Learning is the first release to have both R and Python support.

RELEASE	ANACONDA VERSION	MICROSOFT PACKAGES
SQL Server 2017 Machine Learning Services	4.2 over Python 3.5	revoscalepy, microsoftml

You should never manually overwrite the version of Python installed by SQL Server Setup with newer versions on the web. Microsoft Python packages are based on specific versions of Anaconda. Modifying your installation could destabilize it.

## Component upgrades

After an initial installation, R and Python packages are refreshed through service packs and cumulative updates, but full version upgrades are only possible by *binding* to the Modern Lifecycle Support policy. Binding changes the servicing model. By default, after an initial installation, R packages are refreshed through service packs and cumulative updates. Additional packages and full version upgrades of core R components are only possible through product upgrades (from SQL Server 2016 to SQL Server 2017) or by binding R support to Microsoft Machine Learning Server. For more information, see [Upgrade R and Python components in SQL Server](#).

## Package library location

When you install machine learning with SQL Server, a single package library is created at the instance level for each language that you install. On Windows, the instance library is a secured folder registered with SQL Server.

All script or code that runs in-database on SQL Server must load functions from the instance library. SQL Server cannot access packages installed to other libraries. This applies to remote clients as well. When connecting to the server from a remote client, any R or Python code that you want to run in the server compute context can only

use packages installed in the instance library.

To protect server assets, the default instance library can be modified only by a computer administrator. If you are not the owner of the computer, you might need to get permission from an administrator to install packages to this library.

#### File path for In-Database engine instances

The following table shows the file location of R and Python for version and database engine instance combinations. MSSQL13 indicates SQL Server 2016 and is R-only. MSSQL14 indicates SQL Server 2017 and has R and Python folders.

File paths also include instance names. SQL Server installs [database engine instances](#) as the default instance (MSSQLSERVER) or as a user-defined named instance. If SQL Server is installed as a named instance, you will see that name appended as follows: `MSSQL13.<instance_name>`.

VERSION AND LANGUAGE	DEFAULT PATH
SQL Server 2016	C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library
SQL Server 2017 with R	C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library
SQL Server 2017 with Python	C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\Lib\site-packages

#### File path for Standalone server installations

The following table lists the default paths of the binaries when SQL Server 2016 R Server (Standalone) or SQL Server 2017 Machine Learning Server (Standalone) server is installed.

VERSION	INSTALLATION	DEFAULT PATH
SQL Server 2016	R Server (Standalone)	C:\Program Files\Microsoft SQL Server\130\R_SERVER
SQL Server 2017	Machine Learning Server, with R	C:\Program Files\Microsoft SQL Server\140\R_SERVER
SQL Server 2017	Machine Learning Server, with Python	C:\Program Files\Microsoft SQL Server\140\PYTHON_SERVER

#### NOTE

If you find other folders having similar subfolder names and files, you probably have a standalone installation of Microsoft R Server or [Machine Learning server](#). These server products have different installers and paths (namely, C:\Program Files\Microsoft\R Server\R\_SERVER or C:\Program Files\Microsoft\ML SERVER\R\_SERVER). For more information, see [Install Machine Learning Server for Windows](#) or [Install R Server 9.1 for Windows](#).

## Next steps

- [Get package information](#)
- [Install new R packages](#)
- [Install new Python packages](#)
- [Enable remote R package management](#)

- RevoScaleR functions for R package management
- R package synchronization
- miniCRAN for local R package repository

# Get R and Python package information

6/1/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Sometimes when you are working with multiple environments or installations of R or Python, you need to verify that the code you are running is using the expected environment for Python or the correct workspace for R. For example, if you have upgraded the machine learning components through [binding](#), the path to the R library might be in a different folder than the default. Also, if you install R Client or an instance of the Standalone server, you might have multiple R libraries on your computer.

R and Python script examples in this article show you how to get the path and version of packages used by SQL Server.

## Get the R library location

For any version of SQL Server, run the following statement to verify the [default R package library](#) for the current instance:

```
EXECUTE sp_execute_external_script
 @language = N'R',
 @script = N'OutputDataSet <- data.frame(.libPaths());'
WITH RESULT SETS (([DefaultLibraryName] VARCHAR(MAX) NOT NULL));
GO
```

Optionally, you can use [rxSqlLibPaths](#) in newer versions of RevoScaleR in SQL Server 2017 Machine Learning Services or [R Services upgraded R to at least RevoScaleR 9.0.1](#). This stored procedure returns the path of the instance library and the version of RevoScaleR used by SQL Server:

```
EXECUTE sp_execute_external_script
 @language =N'R',
 @script=N'
sql_r_path <- rxSqlLibPaths("local")
print(sql_r_path)
version_info <-packageVersion("RevoScaleR")
print(version_info)'
```

### NOTE

The [rxSqlLibPaths](#) function can be executed only on the local computer. The function cannot return library paths for remote connections.

## Results

```
STDOUT message(s) from external script:
[1] "C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER1000/R_SERVICES/library"
[1] '9.3.0'
```

## Get the Python library location

For **Python** in SQL Server 2017, run the following statement to verify the default library for the current instance. This example returns the list of folders included in the Python `sys.path` variable. The list includes the current directory, and the standard library path.

```
EXECUTE sp_execute_external_script
@language =N'Python',
@script=N'import sys; print("\n".join(sys.path))'
```

## Results

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\python35.zip
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\DLLs
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\Sphinx-1.5.4-
py3.5.egg
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\win32
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\win32\lib
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\Pythonwin
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\setuptools-27.2.0-
py3.5.egg
```

For more information about the variable `sys.path` and how it is used to set the interpreter's search path for modules, see the [Python documentation](#)

## List all packages

There are multiple ways that you can get a complete list of the packages currently installed. One advantage of running package list commands from `sp_execute_external_script` is that you are guaranteed to get packages installed in the instance library.

### R

The following example uses the R function `installed.packages()` in a Transact-SQL stored procedure to get a matrix of packages that have been installed in the R\_SERVICES library for the current instance. This script returns package name and version fields in the DESCRIPTION file, only the name is returned.

```
EXECUTE sp_execute_external_script
@language=N'R',
@script = N'str(OutputDataSet);
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
OutputDataSet <- data.frame(Name, Version)';
@input_data_1 = N''
WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max)))
```

For more information about the optional and default fields for the R package DESCRIPTION field, see <https://cran.r-project.org>.

### Python

The `pip` module is installed by default, and supports many operations for listing installed packages, in addition to those supported by standard Python. You can run `pip` from a Python command prompt, of course, but you can also call some pip functions from `sp_execute_external_script`.

```

EXECUTE sp_execute_external_script
@language = N'Python',
@script = N'
import pip
import pandas as pd
installed_packages = pip.get_installed_distributions()
installed_packages_list = sorted(["%s==%s" % (i.key, i.version)
 for i in installed_packages])
df = pd.DataFrame(installed_packages_list)
OutputDataSet = df'
WITH RESULT SETS ((PackageVersion nvarchar (150)))

```

When running `pip` from the command line, there are many other useful functions: `pip list` gets all packages that are installed, whereas `pip freeze` lists the packages installed by `pip`, and doesn't list packages that pip itself depends on. You can also use `pip freeze` to generate a dependency file.

## Find a single package

If you have installed a package and want to make sure that it is available to a particular SQL Server instance, you can execute the following stored procedure call to load the package and return only messages.

### R

This example looks for and loads the RevoScaleR library, if available.

```

EXECUTE sp_execute_external_script
@language =N'R',
@script=N'require("RevoScaleR")'
GO

```

- If the package is found, a message is returned: "Commands completed successfully."
- If the package cannot be located or loaded, you get an error containing the text: "there is no package called 'MissingPackageName'"

### Python

The equivalent check for Python can be performed from the Python shell, using `conda` or `pip` commands.

Alternatively, run this statement in a stored procedure:

```

EXECUTE sp_execute_external_script
@language = N'Python',
@script = N'
import pip
import pkg_resources
pckg_name = "revoscalepy"
pckgs = pandas.DataFrame([(i.key) for i in pip.get_installed_distributions()], columns = ["key"])
installed_pckg = pckgs.query('key == @pckg_name')
print("Package", pckg_name, "is", "not" if installed_pckg.empty else "", "installed")

```

## Get package information in R and Python tools

All of the previous instructions assume you are using a SQL Server tool such as Management Studio (SSMS). If you prefer using R and Python tools, the following instructions explain how to get library and package information from an R or Python command line.

### R commands

The instance library for R is \Program Files\Microsoft SQL

Server\MSSQL14.MSSQLSERVER\R\_SERVICES\library.

Launch the standard R tools from these locations to ensure packages from the instance library are used:

- \MSSQL14.MSSQLSERVER\R\_SERVICES\bin\R.exe for an R command prompt
- \MSSQL14.MSSQLSERVER\R\_SERVICES\bin\x64\Rgui.exe for an R console app

You can use standard R commands or RevoScaleR commands to get package information. The RevoScaleR package is loaded automatically.

Return RevoScaleR package information:

```
> print(Revo.version)
```

Return a list of all installed packages:

```
> installed.packages()
```

Return the version of R:

```
> packageDescription("base")
```

## Python commands

Python support is SQL Server 2017 only. The instance library for Python is \Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON\_SERVICES.

Open the Python command window from this location to ensure packages from the instance library are used:

- \MSSQL14.MSSQLSERVER\PYTHON\_SERVICES\Python.exe

Open interactive help:

```
> help()
```

Type a module name at the help prompt to get the package contents, version, and file location:

```
help> revoscalepy
```

## Python package managers (Pip and Conda)

Anaconda includes Python tools for managing packages. On a default instance, Pip, Conda, and other tools can be found at \Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON\_SERVICES\Scripts.

SQL Server Setup does not add Pip or Conda to the system path and on a production SQL Server instance, keeping non-essential executables out of the path is a best practice. However, for development and test environments, you could add the Scripts folder to the system PATH environment variable to run both Pip and Conda on command from any location.

1. Go to C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON\_SERVICES\Scripts
2. Right-click **conda.exe** > **Run as administrator**, and enter `conda list` to return a list of packages installed in the current environment.
3. Similarly, right-click **pip.exe** > **Run as administrator**, and enter `pip list` to return the same information.

## Next steps

- [Install new R packages](#)
- [Install new Python packages](#)
- [Tutorials, samples, solutions](#)

# Install new Python packages on SQL Server

5/16/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to install new Python packages on an instance of SQL Server 2017 Machine Learning Services. In general, the process for installing new packages is similar to that in a standard Python environment. However, some additional steps are required if the server does not have an internet connection.

For help figuring out where packages are installed, or which packages are installed, see [Get R or Python package information](#).

## Prerequisites

- You must have installed SQL Server 2017 Machine Learning Services (In-Database) with the Python language option. For instructions, see [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#).
- For each server instance, you must install a separate copy of the package. Packages cannot be shared across instances.
- Packages must be Python 3.5 compliant and run on Windows.
- Assess whether the package is a good fit for use in the SQL Server environment. Typically a database server supports multiple services and applications, and resources on the file system might be limited, as well as connections to the server. In many cases Internet access is blocked entirely.

Other common problems include the use of networking functionality that is blocked on the server or by the firewall, or packages with dependencies that cannot be installed on a Windows computer.

Some popular Python packages (like Flask) perform tasks such as web development that run better in a standalone environment. We recommend that you use Python in-database for tasks such as machine learning, that require intensive data processing that benefit from tight integration with the database engine, rather than simply querying the database.

- Administrative access to the server is required to install packages.

## Add a new Python package

For this example, we assume that you want to install a new package directly on the SQL Server computer.

The package installed in this example is [CNTK](#), a framework for deep learning from Microsoft that supports customization, training, and sharing of different types of neural networks.

### TIP

Need help configuring your Python tools? See these blogs:

[Getting Started with Python Web Services using Machine Learning Server](#)

[David Crook: Microsoft Cognitive Toolkit + VS Code](#)

### Step 1. Download the Windows version of the Python package

- If you are installing Python packages on a server with no internet access, you must download the WHL file

to a different computer and then copy it to the server.

For example, on a separate computer, you can download the WHL file from this site <https://cntk.ai/PythonWheel/CPU-Only>, and then copy the file `cntk-2.1-cp35-cp35m-win_amd64.whl` to a local folder on the SQL Server computer.

- SQL Server 2017 uses Python 3.5.

#### IMPORTANT

Make sure that you get the Windows version of the package. If the file ends in .gz, it's probably not the right version.

This page contains downloads for multiple platforms and for multiple Python versions: [Set up CNTK](#)

### Step 2. Open a Python command prompt

Locate the default Python library location used by SQL Server. If you have installed multiple instances, locate the PYTHON\_SERVICE folder for the instance where you want to add the package.

For example, if Machine Learning Services has been installed using defaults, and machine learning is enabled on the default instance, the path would be as follows:

```
`C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES`
```

Open the Python command prompt associated with the instance.

#### TIP

For future debugging and testing, you might want to set up a Python environment specific to the instance library.

### Step 3. Install the package using pip

- If you are accustomed to using the Python command line, use PIP.exe to install new packages. You can find the **pip** installer in the `Scripts` subfolder.

SQL Server Setup does not add Scripts to the system path. If you get an error that `pip` is not recognized as an internal or external command, you can add the Scripts folder to the PATH variable in Windows.

The full path of the **Scripts** folder in a default installation is as follows:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\Scripts
```

- If you are using Visual Studio 2017, or Visual Studio 2015 with the Python extensions, you can run `pip install` from the **Python Environments** window. Click **Packages**, and in the text box, provide the name or location of the package to install. You don't need to type `pip install`; it is filled in for you automatically.
  - If the computer has Internet access, provide the name of the package, or the URL of a specific package and version.

For example, to install the version of CNTK that is supported for Windows and Python 3.5, specify the download URL: [https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win\\_amd64.whl](https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win_amd64.whl)

- If the computer does not have internet access, you must download the WHL file before beginning installation. Then, specify the local file path and name. For example, paste the following path and file to install the WHL file downloaded from the site:

```
"C:\Downloads\CNTK\cntk-2.1-cp35-cp35m-win_amd64.whl"
```

You might be prompted to elevate permissions to complete the install.

As installation progresses, you can see status messages in the command prompt window:

```
pip install https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win_amd64.whl
Collecting cntk==2.1 from https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win_amd64.whl
 Downloading https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win_amd64.whl (34.1MB)
 100% |#####| 34.1MB 13kB/s
...
Installing collected packages: cntk
Successfully installed cntk-2.1
```

#### Step 4. Load the package or its functions as part of your script

When installation is complete, you can immediately begin using the package as described in the next step.

For examples of deep learning using CNTK, see these tutorials: [Python API for CNTK](#)

To use functions from the package in your script, insert the standard `import <package_name>` statement in the initial lines of the script:

```
import numpy as np
import cntk as cntk
cntk._version_
```

## List installed packages using conda

There are different ways that you can get a list of installed packages. For example, you can view the installed packages in the **Python Environments** windows of Visual Studio.

If you are using the Python command line, you can use either **Pip** or the **conda** package manager, included with the Anaconda Python environment added by SQL Server setup.

Assuming you added the Scripts folder to the PATH environment variable, run this command from an administrator's command prompt to list the packages in your Python environment. Otherwise, see [Get R and Python package information](#) for pointers on how to run the Python tools in SQL Server.

```
conda list
```

For more information about **conda** and how you can use it to create and manage multiple Python environments, see [Managing environments with conda](#).

# Install new R packages on SQL Server

6/1/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes how to install new R packages to an instance of SQL Server where machine learning is enabled. There are multiple methods for installing new R packages, depending on which version of SQL Server you have, and whether the server has an internet connection. The following approaches for new package installation are possible.

APPROACH	PERMISSIONS	REMOTE/LOCAL
<a href="#">Use conventional R package managers</a>	Admin	Local
<a href="#">Use RevoScaleR</a>	Admin-enabled, database roles afterwards	both
<a href="#">Use T-SQL (CREATE EXTERNAL LIBRARY)</a>	Admin-enabled, database roles afterwards	both

## Who installs (permissions)

The R package library is physically located in the Program Files folder of your SQL Server instance, in a secure folder with restricted access. Writing to this location requires administrator permissions.

Non-administrators can install packages but doing so requires additional configuration and capability not available in initial installations. There are two approaches for non-admin package installations: RevoScaleR using version 9.0.1 and later, or using CREATE EXTERNAL LIBRARY (SQL Server 2017 only). In SQL Server 2017, **dbo\_owner** or another user with CREATE EXTERNAL LIBRARY permission can install R packages to the current database.

R developers are accustomed to creating user libraries for the packages they need if centrally located libraries are off-limits. This practice is problematic for R code executing in a SQL Server database engine instance. SQL Server cannot load packages from external libraries, even if that library is on the same computer. Only packages from the instance library can be used in R code running in SQL Server.

File system access is typically restricted on the server, and even if you have admin rights and access to a user document folder on the server, the external script runtime that executes in SQL Server cannot access any packages installed outside the default instance library.

## Considerations for package installation

Before installing new packages, consider whether the capabilities enabled by a given package are suitable in a SQL Server environment. On a hardened SQL Server environment, you might want to avoid the following:

- Packages that require network access
- Packages that require Java or other frameworks not typically used in a SQL Server environment
- Packages that require elevated file system access
- Package is used for web development or other tasks that don't benefit by running inside SQL Server

## Offline installation (no internet access)

In general, servers that host production databases block internet connections. Installing new R or Python packages in such environments requires that you prepare packages and dependencies in advance, and copy the files to a folder on the server for offline installation.

Identifying all dependencies gets complicated. For R, we recommend that you use [miniCRAN to create a local repository](#) and then transfer the fully defined repo to an isolated SQL Server instance.

Alternativley, you can perform this steps manually:

1. Identify all package dependencies.
2. Check whether any required packages are already installed on the server. If the package is installed, verify that the version is correct.
3. Download the package and all dependencies to a separate computer.
4. Move the files to a folder accessible by the server.
5. Run a supported installation command or DDL statement to install the package into the instance library.

### Download the package as a zipped file

For installation on a server without internet access, you must download a copy of the package in the format of a zipped file for offline installation. **Do not unzip the package.**

For example, the following procedure describes how to get the correct version of the [FISHalyseR](#) package from Bioconductor, assuming the computer has access to the internet.

1. In the **Package Archives** list, find the **Windows binary** version.
  2. Right-click the link to the .ZIP file, and select **Save target as...**.
  3. Navigate to the local folder where zipped packages are stored, and click **Save**.
- This process creates a local copy of the package.
4. If you get a download error, try a different mirror site.
  5. After the package archive has been downloaded, you can install the package, or copy the zipped package to a server that does not have internet access.

#### TIP

If by mistake you install the package instead of downloading the binaries, a copy of the downloaded zipped file is also saved to your computer. Watch the status messages as the package is installed to determine the file location. You can copy that zipped file to the server that does not have internet access.

However, when you obtain packages using this method, the dependencies are not included.

## Side-by-side installation with Standalone R or Python Servers

R and Python features are included in several Microsoft products, all of which could co-exist on the same computer.

If you installed SQL Server 2017 Microsoft Machine Learning Server (Standalone) or SQL Server 2016 R Server (Standalone) in addition to in-database analytics (SQL Server 2017 Machine Learning Services and SQL Server 2016 R Services), your computer has separate installations of R for each, with duplicates of all the R tools and libraries.

Packages that are installed to the R\_SERVER library are used only by a standalone server and cannot be

accessed by a SQL Server (In-Database) instance. Always use the `R_SERVICES` library when installing packages that you want to use in-database on SQL Server. For more information about paths, see [Package library location](#).

## See also

- [Install new Python packages](#)
- [Tutorials, samples, solutions](#)

# Use R package managers to install R packages on SQL Server

6/1/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

You can use standard R tools to install new packages on an instance of SQL Server 2016 or SQL Server 2017, providing the computer has an open port 80 and you have administrator rights.

## IMPORTANT

Be sure to install packages to the default library that is associated with the current instance. Never install packages to a user directory.

This procedure uses RGui but you can use RTerm or any other R command-line tool that supports elevated access.

## Install a package using RGui

1. [Determine the location of the instance library](#). Navigate to the folder where the R tools are installed. For example, the default path for a SQL Server 2017 default instance is as follows:  
`C:\Program Files\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64`
2. Right-click RGui.exe, and select **Run as administrator**. If you do not have the required permissions, contact the database administrator and provide a list of the packages you need.
3. From the command line, if you know the package name, you can type:  
`install.packages("the_package-name")` Double quotation marks are required for the package name.
4. When asked for a mirror site, select any site that is convenient for your location.

If the target package depends on additional packages, the R installer automatically downloads the dependencies and installs them for you.

If you have multiple instances of SQL Server, such as side-by-side instances of SQL Server 2016 R Services and SQL Server 2017 Machine Learning Services, run installation separately for each instance if you want to use the package in both contexts. Packages cannot be shared across instances.

## Offline installation using R tools

If the server does not have internet access, additional steps are required to prepare the packages. To install R packages on a server that does not have internet access, you must:

- Analyze dependencies in advance.
- Download the target package to a computer with Internet access.
- Download any required packages to the same computer and place all packages in a single package archive.
- Zip the archive if it is not already in zipped format.
- Copy the package archive to a location on the server.
- Install the target package specifying the archive file as source.

## IMPORTANT

Be sure that you analyze all dependencies and download **all** required packages **before** beginning installation. We recommend [miniCRAN](#) for this process. This R package takes a list of packages you want to install, analyzes dependencies, and gets all the zipped files for you. miniCRAN then creates a single repository that you can copy to the server computer. For details, see [Create a local package repository using miniCRAN](#)

This procedure assumes that you have prepared all the packages that you need, in zipped format, and are ready to copy them to the server.

1. Copy the package zipped file, or for multiple packages, the complete repository containing all packages in zipped format, to a location that the server can access.
2. Open the folder on the server where the R tools for the instance are installed. For example, if you are using the Windows command prompt on a system with SQL Server 2016 R Services, switch to `C:\Program Files\MSSQL13.MSSQLSERVER\R_SERVICES\bin\x64`.
3. Right-click on RGui or RTerm and select **Run as administrator**.
4. Run the R command `install.packages` and specify the package or repository name, and the location of the zipped files.

```
install.packages("C:\\Temp\\Downloaded packages\\mynewpackage.zip", repos=NULL)
```

This command extracts the R package `mynewpackage` from its local zipped file, assuming you saved the copy in the directory `C:\\Temp\\Downloaded packages`, and installs the package on the local computer. If the package has any dependencies, the installer checks for existing packages in the library. If you have created a repository that includes the dependencies, the installer installs the required packages as well.

If any required packages are not present in the instance library, and cannot be found in the zipped files, installation of the target package fails.

## See also

- [Install new R packages](#)
- [Install new Python packages](#)
- [Tutorials, samples, solutions](#)

# Use T-SQL (CREATE EXTERNAL LIBRARY) to install R packages on SQL Server 2017 Machine Learning Services

5/30/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article explains how to install new R packages on an instance of SQL Server where machine learning is enabled. There are multiple approaches to choose from. Using T-SQL works best for server administrators who are unfamiliar with R.

**Applies to:** SQL Server 2017 (14.x) Machine Learning Services (In-Database)

The [CREATE EXTERNAL LIBRARY](#) statement makes it possible to add a package or set of packages to an instance or a specific database without running R or Python code directly. However, this method requires package preparation and additional database permissions.

- All packages must be available as a local zipped file, rather than downloaded on demand from the internet.
- All dependencies must be identified by name and version, and included in the zip file. The statement fails if required packages are not available, including downstream package dependencies.
- You must be **db\_owner** or have CREATE EXTERNAL LIBRARY permission in a database role. For details, see [CREATE EXTERNAL LIBRARY](#).

## Download packages in archive format

If you are installing a single package, download the package in zipped format.

It's more common to install multiple packages due to package dependencies. When a package requires other packages, you must verify that all of them are accessible to each other during installation. We recommend [creating a local repository](#) using [miniCRAN](#) to assemble a full collection of packages, as well as [igraph](#) for analyzing packages dependencies. Installing the wrong version of a package or omitting a package dependency can cause a CREATE EXTERNAL LIBRARY statement to fail.

## Copy the file to a local folder

Copy the zipped file containing all packages to a local folder on the server. If you do not have access to the file system on the server, you can also pass a complete package as a variable, using a binary format. For more information, see [CREATE EXTERNAL LIBRARY](#).

## Run the statement to upload packages

Open a **Query** window, using an account with administrative privileges.

Run the T-SQL statement `CREATE EXTERNAL LIBRARY` to upload the zipped package collection to the database.

For example, the following statement names as the package source a miniCRAN repository containing the **randomForest** package, together with its dependencies.

```
CREATE EXTERNAL LIBRARY randomForest
FROM (CONTENT = 'C:\Temp\Rpackages\randomForest_4.6-12.zip')
WITH (LANGUAGE = 'R');
```

You cannot use an arbitrary name; the external library name must have the same name that you expect to use when loading or calling the package.

## Verify package installation

If the library is successfully created, you can run the package in SQL Server, by calling it inside a stored procedure.

```
EXEC sp_execute_external_script
@language =N'R',
@script=N'library(randomForest)'
```

## See also

- [Get package information](#)
- [R tutorials](#)
- [How-to guides](#)

# How to use RevoScaleR functions to find or install R packages on SQL Server

6/1/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

RevoScaleR 9.0.1 and later includes functions for R package management a SQL Server compute context. These functions can be used by remote, non-administrators to install packages on SQL Server without direct access to the server.

SQL Server 2017 Machine Learning Services already includes a newer version of RevoScaleR. SQL Server 2016 R Services customers must do a [component upgrade](#) to get RevoScaleR package management functions. For instructions on how to retrieve package version and contents, see [Get package information](#).

## RevoScaleR functions for package management

The following table describes the functions used for R package installation and management.

FUNCTION	DESCRIPTION
<a href="#">rxSqlLibPaths</a>	Determine the path of the instance library on the remote SQL Server.
<a href="#">rxFindPackage</a>	Gets the path for one or more packages on the remote SQL Server.
<a href="#">rxInstallPackages</a>	Call this function from a remote R client to install packages in a SQL Server compute context, either from a specified repository, or by reading locally saved zipped packages. This function checks for dependencies and ensures that any related packages can be installed to SQL Server, just like R package installation in the local compute context. To use this option, you must have enabled package management on the server and database. Both client and server environments must have the same version of RevoScaleR.
<a href="#">rxInstalledPackages</a>	Gets a list of packages installed in the specified compute context.
<a href="#">rxSyncPackages</a>	Copy information about a package library between the file system and database, for the specified compute context.
<a href="#">rxRemovePackages</a>	Removes packages from a specified compute context. It also computes dependencies and ensures that packages that are no longer used by other packages on SQL Server are removed, to free up resources.

## Prerequisites

- [Enable remote R package management on SQL Server](#)
- RevoScaleR versions must be the same on both client and server environments. For more information, see

## Get package information.

- Permission to connect to the server and a database, and to run R commands. You must be a member of a database role that allows you to install packages on the specified instance and database.
- Packages in **shared scope** can be installed by users belonging to the `rpkgs-shared` role in a specified database. All users in this role can uninstall shared packages.
- Packages in **private scope** can be installed by any user belonging to the `rpkgs-private` role in a database. However, users can see and uninstall only their own packages.
- Database owners can work with shared or private packages.

## Client connections

A client workstation can be [Microsoft R Client](#) or a [Microsoft Machine Learning Server](#) (data scientists often use the free developer edition) on the same network.

When calling package management functions from a remote R client, you must create a compute context object first, using the `RxInSqlServer` function. Thereafter, for each package management function that you use, pass the compute context as an argument.

User identity is typically specified when setting the compute context. If you do not specify a user name and password when you create the compute context, the identity of the user running the R code is used.

1. From an R command line, define a connection string to the instance and database.
2. Use the `RxInSqlServer` constructor to define a SQL Server compute context, using the connection string.

```
sqlcc <- RxInSqlServer(connectionString = myConnString, shareDir = sqlShareDir, wait = sqlWait,
consoleOutput = sqlConsoleOutput)
```

3. Create a list of the packages you want to install and save the list in a string variable.

```
packageList <- c("e1071", "mice")
```

4. Call `rxInstallPackages` and pass the compute context and the string variable containing the package names.

```
rxInstallPackages(pkgs = packageList, verbose = TRUE, computeContext = sqlcc)
```

If dependent packages are required, they are also installed, assuming an internet connection is available on the client.

Packages are installed using the credentials of the user making the connection, in the default scope for that user.

## Call package management functions in stored procedures

You can run package management functions inside `sp_execute_external_script`. When you do so, the function is executed using the security context of the stored procedure caller.

## Examples

This section provides examples of how to use these functions from a remote client when connecting to a SQL Server instance or database as the compute context.

For all examples, you must provide either a connection string, or a compute context, which requires a connection string. This example provides one way to create a compute context for SQL Server:

```
instance_name <- "computer-name(instance-name)";
database_name <- "TestDB";
sqlWait= TRUE;
sqlConsoleOutput <- TRUE;
connString <- paste("Driver=SQL Server;Server=", instance_name, ";Database=", database_name,
";Trusted_Connection=true;", sep="");
sqlcc <- RxInSqlServer(connectionString = connString, wait = sqlWait, consoleOutput = sqlConsoleOutput,
numTasks = 4);
```

Depending on where the server is located, and the security model, you might need to provide a domain and subnet specification in the connection string, or use a SQL login. For example:

```
connStr <- "Driver=SQL
Server;Server=myserver.financeweb.contoso.com;Database=Finance;Uid=RUser1;Pwd=RUserPassword"
```

### Get package path on a remote SQL Server compute context

This example gets the path for the **RevoScaleR** package on the compute context, `sqlcc`.

```
sqlPackagePaths <- rxFindPackage(package = "RevoScaleR", computeContext = sqlcc)
print(sqlPackagePaths)
```

### Results

"C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R\_SERVICES/library/RevoScaleR"

#### TIP

If you have enabled the option to see SQL console output, you might get status messages from the function that precedes the `print` statement. After you have finished testing your code, set `consoleOutput` to FALSE in the compute context constructor to eliminate messages.

### Get locations for multiple packages

The following example gets the paths for the **RevoScaleR** and **lattice** packages, on the compute context, `sqlcc`. To get information about multiple packages, pass a string vector containing the package names.

```
packagePaths <- rxFindPackage(package = c("RevoScaleR", "lattice"), computeContext = sqlcc)
print(packagePaths)
```

### Get package versions on a remote compute context

Run this command from an R console to get the build number and version numbers for packages installed on the compute context, `sqlServer`.

```
sqlPackages <- rxInstalledPackages(fields = c("Package", "Version", "Built"), computeContext = sqlServer)
```

### Results

```
[1] "C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library/RevoScaleR"
```

```
[2] "C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library/lattice"
```

## Install a package on SQL Server

This example installs the **forecast** package and its dependencies into the compute context.

```
pkgs <- c("forecast")
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)
```

## Remove a package from SQL Server

This example removes the **forecast** package and its dependencies from the compute context.

```
pkgs <- c("forecast")
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)
```

## Synchronize packages between database and file system

The following example checks the database **TestDB**, and determines whether all packages are installed in the file system. If some packages are missing, they are installed in the file system.

```
Instantiate the compute context
connectionString <- "Driver=SQL Server;Server=myServer;Database=TestDB;Trusted_Connection=True;"
computeContext <- RxInSqlServer(connectionString = connectionString)

Synchronize the packages in the file system for all scopes and users
rxSyncPackages(computeContext=computeContext, verbose=TRUE)
```

Package synchronization works on a per database and per user basis. For more information, see [R package synchronization for SQL Server](#).

## Use a stored procedure to list packages in SQL Server

Run this command from Management Studio or another tool that supports T-SQL, to get a list of installed packages on the current instance, using `rxInstalledPackages` in a stored procedure.

```
EXEC sp_execute_external_script
 @language=N'R',
 @script=N'
 myPackages <- rxInstalledPackages();
 OutputDataSet <- as.data.frame(myPackages);
 '
```

The `rxSqlLibPaths` function can be used to determine the active library used by SQL Server Machine Learning Services. This script can return only the library path for the current server.

```
declare @instance_name nvarchar(100) = @@SERVERNAME, @database_name nvarchar(128) = db_name();
exec sp_execute_external_script
 @language = N'R',
 @script = N'
connStr <- paste("Driver=SQL Server;Server=", instance_name, ";Database=", database_name,
";Trusted_Connection=true;", sep="");
.libPaths(rxSqlLibPaths(connStr));
print(.libPaths());
',
@input_data_1 = N'',
@params = N'@instance_name nvarchar(100), @database_name nvarchar(128)',
@instance_name = @instance_name,
@database_name = @database_name;
```

## See also

- [Enable remote R package management](#)
- [Synchronize R packages](#)
- [Tips for installing R packages](#)
- [Default packages](#)

# Enable or disable remote package management for SQL Server

6/1/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article describes how to enable remote management of R packages from a client workstation or a different Machine Learning Server. After the package management feature has been enabled on SQL Server, you can use RevoScaleR commands on a client to install packages on SQL Server.

## NOTE

Currently management of R libraries is supported; support for Python is on the roadmap.

By default, the external package management feature for SQL Server is disabled. You must run a separate script to enable the feature as described in the next section.

## Overview of process and tools

To enable or disable package management on SQL Server, use the command-line utility **RegisterRExt.exe**, which is included with the **RevoScaleR** package.

**Enabling** this feature is a two-step process, requiring a database administrator: you enable package management on the SQL Server instance (once per SQL Server instance), and then enable package management on the SQL database (once per SQL Server database).

**Disabling** the package management feature also requires multipel steps: you remove database-level packages and permissions (once per database), and then remove the roles from the server (once per instance).

## Enable package management

1. On SQL Server, open an elevated command prompt and navigate to the folder containing the utility, RegisterRExt.exe. The default location is

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterREx.exe .
```

2. Run the following command, providing appropriate arguments for your environment:

```
RegisterRExt.exe /installpkgmgmt [/instance:name] [/user:username] [/password:*|password]
```

This command creates instance-level objects on the SQL Server computer that are required for package management. It also restarts the Launchpad for the instance.

If you do not specify an instance, the default instance is used. If you do not specify a user, the current security context is used. For example, the following command enables package management on the instance in the path of RegisterRExt.exe, using the credentials of the user who opened the command prompt:

```
REregisterRExt.exe /installpkgmgmt
```

3. To add package management to a specific database, run the following command from an elevated command prompt:

```
RegisterRExt.exe /installpkgmgmt /database:databasefilename [/instance:instance] [/user:username]
[/password:*|password]
```

This command creates some database artifacts, including the following database roles that are used for controlling user permissions: `rpkgs-users`, `rpkgs-private`, and `rpkgs-shared`.

For example, the following command enables package management on the database, on the instance where RegisterRExt is run. If you do not specify a user, the current security context is used.

```
RegisterRExt.exe /installpkgmgmt /database:TestDB
```

4. Repeat the command for each database where packages must be installed.
5. To verify that the new roles have been successfully created, in SQL Server Management Studio, click the database, expand **Security**, and expand **Database Roles**.

You can also run a query on `sys.database_principals` such as the following:

```
SELECT pr.principal_id, pr.name, pr.type_desc,
 pr.authentication_type_desc, pe.state_desc,
 pe.permission_name, s.name + '.' + o.name AS ObjectName
FROM sys.database_principals AS pr
JOIN sys.database_permissions AS pe
 ON pe.grantee_principal_id = pr.principal_id
JOIN sys.objects AS o
 ON pe.major_id = o.object_id
JOIN sys.schemas AS s
 ON o.schema_id = s.schema_id;
```

After you have enabled this feature, you can use RevoScaleR function to install or uninstall packages from a remote R client.

## Disable package management

1. From an elevated command prompt, run the RegisterRExt utility again, and disable package management at the database level:

```
RegisterRExt.exe /uninstallpkgmgmt /database:databasefilename [/instance:instance] [/user:username]
[/password:*|password]
```

This command removes database objects related to package management from the specified database. It also removes all the packages that were installed from the secured file system location on the SQL Server computer.

2. Repeat this command on each database where package management was used.
3. (Optional) After all databases have been cleared of packages using the preceding step, run the following command from an elevated command prompt:

```
RegisterRExt.exe /uninstallpkgmgmt [/instance:instance] [/user:username] [/password:*|password]
```

This command removes the package management feature from the instance. You might need to manually restart the Launchpad service once more to see changes.

## Next steps

- [Use RevoScaleR to install new R packages](#)
- [Tips for installing R packages](#)
- [Default packages](#)

# R package synchronization for SQL Server

5/30/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The version of RevoScaleR included in SQL Server 2017 includes the ability to synchronize collections of R packages between the file system and the instance and database where packages are used.

This feature was provided to make it easier to back up R package collections associated with SQL Server databases. Using this feature, an administrator can restore not just the database, but any R packages that were used by data scientists working in that database.

This article describes the package synchronization feature, and how to use the `rxSyncPackages` function to perform the following tasks:

- Synchronize a list of packages for an entire SQL Server database
- Synchronize packages used by an individual user, or by a group of users
- If a user moves to a different SQL Server, you can take a backup of the user's working database and restore it to the new server, and the packages for the user will be installed into the file system on the new server, as required by R.

For example, you might use package synchronization in these scenarios:

- The DBA has restored an instance of SQL Server to a new machine and asks users to connect from their R clients and run `rxSyncPackages` to refresh and restore their packages.
- You think an R package on the file system is corrupted so you run `rxSyncPackages` on the SQL Server.

## Requirements

Before you can use package synchronization, you must have the appropriate version of Microsoft R or Machine Learning Server. This feature is provided in Microsoft R version 9.1.0 or later.

You must also enable the [package management feature](#) on the server.

### Determine whether your server supports package management

This feature is available in SQL Server 2017 CTP 2 or later.

You can add this feature to an instance of SQL Server 2016 by upgrading the instance to use the latest version of Microsoft R. For more information, see [Use SqlBindR.exe to upgrade SQL Server R Services](#).

### Enable the package management feature

To use package synchronization requires that the new package management feature be enabled on the SQL Server instance, and on individual databases. For more information, see [Enable or disable package management for SQL Server](#).

1. The server administrator enables the feature for the SQL Server instance.
2. For each database, the administrator grants individual users the ability to install or share R packages, using database roles.

When this is done, you can use RevoScaleR functions, such as `rxInstallPackages` to install packages into a database. Information about users and the packages that they can use is stored in the SQL Server instance.

Whenever you add a new package using the package management functions, both the records in SQL Server and the file system are updated. This information can be used to restore package information for the entire database.

## Permissions

- The person who executes the package synchronization function must be a security principal on the SQL Server instance and database that has the packages.
- The caller of the function must be a member of one of these package management roles: **rpkgs-shared** or **rpkgs-private**.
- To synchronize packages marked as **shared**, the person who is running the function must have membership in the **rpkgs-shared** role, and the packages that are being moved must have been installed to a shared scope library.
- To synchronize packages marked as **private**, either the owner of the package or the administrator must run the function, and the packages must be private.
- To synchronize packages on behalf of other users, the owner must be a member of the **db\_owner** database role.

## How package synchronization works

To use package synchronization, call `rxSyncPackages`, which is a new function in [RevoScaleR](#).

For each call to `rxSyncPackages`, you must specify a SQL Server instance and database. Then, either list the packages to synchronize, or specify package scope.

1. Create the SQL Server compute context by using the `RxInSqlServer` function. If you don't specify a compute context, the current compute context is used.
2. Provide the name of a database on the instance in the specified compute context. Packages are synchronized per database.
3. Specify the packages to synchronize by using the scope argument.

If you use **private** scope, only packages owned by the specified owner are synchronized. If you specify **shared** scope, all non-private packages in the database are synchronized.

If you run the function without specifying either **private** or **shared** scope, all packages are synchronized.

4. If the command is successful, existing packages in the file system are added to the database, with the specified scope and owner.

If the file system is corrupted, the packages are restored based on the list maintained in the database.

If the package management feature is not available on the target database, an error is raised: "The package management feature is either not enabled on the SQL Server or version is too old"

### Example 1. Synchronize all package by database

This example gets any new packages from the local file system and installs the packages in the database [TestDB]. Because no owner is specific, the list includes all packages that have been installed for private and shared scopes.

```
connectionString <- "Driver=SQL Server;Server=myServer;Database=TestDB;Trusted_Connection=True;"
computeContext <- RxInSqlServer(connectionString = connectionString)
rxSyncPackages(computeContext=computeContext, verbose=TRUE)
```

### Example 2. Restrict synchronized packages by scope

The following examples synchronize only the packages in the specified scope.

```
#Shared scope
rxSyncPackages(computeContext=computeContext, scope="shared", verbose=TRUE)

#Private scope
rxSyncPackages(computeContext=computeContext, scope="private", verbose=TRUE)
```

### Example 3. Restrict synchronized packages by owner

The following example demonstrates how to synchronize only the packages that were installed for a specific user. In this example, the user is identified by the SQL login name, *user1*.

```
rxSyncPackages(computeContext=computeContext, scope="private", owner = "user1", verbose=TRUE)
```

## Related resources

[R package management for SQL Server](#)

# Create a local R package repository using miniCRAN

5/30/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

miniCRAN package, created by [Andre de Vries](#), identifies and downloads packages and dependencies into a single folder, which you can copy to other computers for offline R package installation.

As an input, specify one or more packages. **miniCRAN** recursively reads the dependency tree for these packages and downloads only the listed packages and their dependencies from CRAN or similar repositories.

As an output, **miniCRAN** creates an internally consistent repository consisting of the selected packages and all required dependencies. You can then move this local repository to the server, and proceed to install the packages without an internet connection.

## NOTE

Experienced R users often look for the list of dependent packages in the DESCRIPTION file for the downloaded package. However, packages listed in **Imports** might have second-level dependencies. For this reason, we recommend **miniCRAN** for assembling the full collection of required packages.

## Why create a local repository

The goal of creating a local package repository is to provide a single location that a server administrator or other users in the organization can use to install new R packages on a server, especially one that does not have internet access. After creating the repository, you can modify it by adding new packages or upgrading the version of existing packages.

Package repositories are useful in these scenarios:

- **Security:** Many R users are accustomed to downloading and installing new R packages at will, from CRAN or one of its mirror sites. However, for security reasons, production servers running SQL Server typically do not have internet connectivity.
- **Easier offline installation:** To install package to an offline server requires that you also download all package dependencies. Using miniCRAN makes it easier to get all dependencies in the correct format.

By using miniCRAN, you can avoid package dependency errors when preparing packages to install with the [CREATE EXTERNAL LIBRARY](#) statement.

- **Improved version management:** In a multiuser environment, there are good reasons to avoid unrestricted installation of multiple package versions on the server. Use a local repository to provide a consistent set of packages for use by your analysts.

## TIP

You can also use miniCRAN to prepare packages for use in Azure Machine Learning. For more information, see this blog: [Using miniCRAN in Azure ML, by Michele Usuelli](#)

## Install miniCRAN

The **miniCRAN** package itself is dependent on 18 other CRAN packages, among which is the **RCurl** package, which has a system dependency on the **curl-devel** package. Similarly, package **XML** has a dependency on **libxml2-devel**. To resolve dependencies, we recommend that you build your local repository initially on a machine with full internet access.

Run the following commands on a computer with a base R, R tools, and internet connection. It's assumed that this is *not* your SQL Server computer. The following commands install the **miniCRAN** package and the required **igraph** package. This example checks for whether the package is already installed, but you can bypass the if statements and install the packages directly.

```
if(!require("miniCRAN")) install.packages("miniCRAN")
if(!require("igraph")) install.packages("igraph")
library("miniCRAN")
```

## Set the CRAN mirror and MRAN snapshot

Specify a mirror site to use in getting packages. For example, you could use the MRAN site, or any other site in your region that contains the packages you need. If download fails, try another mirror site.

```
CRAN_mirror <- c(CRAN = "https://mran.microsoft.com")
CRAN_mirror <- c(CRAN = "https://cran.cnr.berkeley.edu")
```

## Create a local folder

Create a local folder, such as `C:\mylocalrepo` in which to store the collected packages. If you repeat this often, you should probably use a more descriptive name, such as "miniCRANZooPackages" or "miniCRANMyRPackagev2".

Specify the folder as the local repo. R syntax uses a forward slash for path names, which is opposite from Windows conventions.

```
local_repo <- "C:/mylocalrepo"
```

## Add packages to the local repo

After **miniCRAN** is installed and loaded, create a list that specifies the additional packages you want to download.

Do **not** add dependencies to this initial list. The **igraph** package used by **miniCRAN** generates the list of dependencies for you. For more information about how to use the generated dependency graph, see [Using miniCRAN to identify package dependencies](#).

1. Add target packages, "zoo" and "forecast" to a variable.

```
pkgs_needed <- c("zoo", "forecast")
```

2. Optionally, plot the dependency graph, but it can be informative.

```
plot(makeDepGraph(pkgs_needed))
```

3. Create the local repo. Be sure to change the R version if necessary to the version installed on your SQL Server instance. Version 3.2.2 is on SQL Server 2016, version 3.3 is on SQL Server 2017. If you did a component upgrade, your version might be newer. For more information, see [Get R and Python package](#)

information.

```
pkgs_expanded <- pkgDep(pkgs_needed, repos = CRAN_mirror);
makeRepo(pkgs_expanded, path = local_repo, repos = CRAN_mirror, type = "win.binary", Rversion = "3.3");
```

From this information, the miniCRAN package creates the folder structure that you need to copy the packages to the SQL Server later.

At this point you should have a folder containing the packages you needed, and any additional packages that were required. The path should be similar to this example: C:\mylocalrepo\bin\windows\contrib\3.3 and it should contain a collection of zipped packages. Do not unzip the packages or rename any files.

Optionally, run the following code to list the packages contained in the local miniCRAN repository.

```
pdb <- as.data.frame(pkgAvail(local_repo, type = "win.binary", Rversion = "3.3"), stringsAsFactors = FALSE);
head(pdb);
pdb$Package;
pdb[, c("Package", "Version", "License")]
```

## Add packages to the instance library

After you have a local repository with the packages you need, move the package repository to the SQL Server computer. The following procedure describes how to install the packages using R tools.

1. Copy the folder containing the miniCRAN repository, in its entirety, to the server where you plan to install the packages. The folder typically has this structure: miniCRAN root> bin > windows > contrib > version > all packages. In the following examples, we assume a folder off the root drive:
2. Open an R tool associated with the instance (for example, you could use Rgui.exe). Right-click **Run as administrator** to allow the tool to make updates to your system.
  - For SQL Server 2017, the file location for RGUI is  
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R\_SERVICES\bin\x64 .
  - For SQL Server 2016, he file location for RGUI is  
C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R\_SERVICES\bin\x64 .
3. Get the path for the instance library, and add it to the list of library paths. On SQL Server 2017, the path is similar to the following example.

```
outputlib <- "C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library"
```

4. Specify the new location on the server where you copied the **miniCRAN** repository, as `server_repo`.

In this example, we assume that you copied the repository to a temporary folder on the server.

```
inputlib <- "C:/temp/mylocalrepo"
```

5. Since you are working in a new R workspace on the server, you must also furnish the list of packages to install.

```
mypackages <- c("zoo", "forecast")
```

6. Install the packages, providing the path to the local copy of the miniCRAN repo.

```
install.packages(mypackages, repos = file.path("file://", normalizePath(inputlib, winslash = "/")), lib = outputlib, type = "win.binary", dependencies = TRUE);
```

7. From the instance library, you can view the installed packages using a command like the following:

```
installed.packages()
```

## See also

- [Get package information](#)
- [R tutorials](#)
- [How-to guides](#)

# Tips for using R packages in SQL Server

6/1/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article has separate sections for DBAs who are unfamiliar with R and experienced R developers who are unfamiliar package access in a SQL Server instance.

## New to R

As an administrator installing R packages for the first time, knowing a few basics about R package management can help you get started.

### Package dependencies

R packages frequently depend on multiple other packages, some of which might not be available in the default R library used by the instance. Sometimes a package requires a different version of a dependent package that is already installed. Package dependencies are noted in a DESCRIPTION file embedded in the package, but are sometimes incomplete. You can use a package called [iGraph](#) to fully articulate the dependency graph.

If you need to install multiple packages, or want to ensure that everyone in your organization gets the correct package type and version, we recommend that you use the [miniCRAN](#) package to analyze the complete dependency chain. minicRAN creates a local repository that can be shared among multiple users or computers. For more information, see [Create a local package repository using miniCRAN](#).

### Package sources, versions, and formats

There are multiple sources for R packages, such as [CRAN](#) and [Bioconductor](#). The official site for the R language (<https://www.r-project.org/>) lists many of these resources. Microsoft offers [MRAN](#) for its distribution of open-source R ([MRO](#)) and other packages. Many packages are published to GitHub, where developers can obtain the source code.

R packages run on multiple computing platforms. Be sure that the versions you install are Windows binaries.

### Know which library you are installing to and which packages are already installed.

If you have previously modified the R environment on the computer, before installing anything, ensure that the R environment variable `.libPath` uses just one path.

This path should point to the R\_SERVICES folder for the instance. For more information, including how to determine which packages are already installed, see [Default R and Python packages in SQL Server](#).

## New to SQL Server

As an R developer working on code executing on SQL Server, the security policies protecting the server constrain your ability to control the R environment.

### R user libraries: not supported on SQL Server

R developers who need to install new R packages are accustomed to installing packages at will, using a private, user library whenever the default library is not available, or when the developer is not an administrator on the computer. For example, in a typical R development environment, the user would add the location of the package to the R environment variable `libPath`, or reference the full package path, like this:

```
library("c:/Users/<username>/R/win-library/packagename")
```

This does not work when running R solutions in SQL Server, because R packages must be installed to a specific default library that is associated with the instance. When a package is not available in the default library, you get this error when you try to call the package:

*Error in library(xxx) : there is no package called 'package-name'*

### Avoid "package not found" errors

- Eliminate dependencies on user libraries.

It is a bad development practice to install required R packages to a custom user library, as it can lead to errors if a solution is run by another user who does not have access to the library location.

Also, if a package is installed in the default library, the R runtime loads the package from the default library, even if you specify a different version in the R code.

- Modify your code to run in a shared environment.
- Avoid installing packages as part of a solution. If you don't have permissions to install packages, the code will fail. Even if you do have permissions to install packages, you should do so separately from other code that you want to execute.
- Check your code to make sure that there are no calls to uninstalled packages.
- Update your code to remove direct references to the paths of R packages or R libraries.
- Know which package library is associated with the instance. For more information, see [Default R and Python packages in SQL Server](#).

## See also

- [Install new R packages](#)
- [Install new Python packages](#)
- [Tutorials, samples, solutions](#)

# R libraries and R data types

5/30/2018 • 7 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This topic describes the R libraries that are included and the data types that are supported in the following products:

- SQL Server 2016 R Services (In-Database)
- SQL Server Machine Learning Services (In-Database)

This topic also lists unsupported data types, and lists the data type conversions that might be performed implicitly when data is passed between R and SQL Server.

## R Libraries

Both products, R Services and Machine Learning Services with R, are aligned with specific releases of Microsoft R Open. For example, the latest release, SQL Server 2017 Machine Learning Services, is built on Microsoft R Open 3.3.3.

To view the R version associated with a particular instance of SQL Server, open RGui.

1. For the default instance, the path would be as follows:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64\
```

2. A message is displayed that lists the R distribution and the Microsoft R Open version number.

To find the version of R included in a particular version of Microsoft R Server, see [R Server - What's New](#).

Note that the package management system in SQL Server means that multiple versions of an R package can be installed on the same computer, with multiple users sharing the same package, or using different versions of the same package. For more information, see [R Package Management in SQL Server](#).

## R and SQL Data Types

Whereas SQL Server supports several dozen data types, R has a limited number of scalar data types (numeric, integer, complex, logical, character, date/time and raw). As a result, whenever you use data from SQL Server in R scripts, data might be implicitly converted to a compatible data type. However, often an exact conversion cannot be performed automatically, and an error is returned, such as "Unhandled SQL data type".

This section lists the implicit conversions that are provided, and lists unsupported data types. Some guidance is provided for mapping data types between R and SQL Server.

## Implicit data type conversions between R and SQL Server

The following table shows the changes in data types and values when data from SQL Server is used in an R script and then returned to SQL Server.

SQL TYPE	R CLASS	RESULT SET TYPE	COMMENTS
<b>bigint</b>	numeric	float	

SQL TYPE	R CLASS	RESULT SET TYPE	COMMENTS
<b>binary(n)</b> n <= 8000	raw	<b>varbinary(max)</b>	Only allowed as input parameter and output
<b>bit</b>	logical	<b>bit</b>	
<b>char(n)</b> n <= 8000	character	<b>varchar(max)</b>	
<b>datetime</b>	POSIXct	<b>datetime</b>	Represented as GMT
<b>date</b>	POSIXct	<b>datetime</b>	Represented as GMT
<b>decimal(p,s)</b>	numeric	<b>float</b>	
<b>float</b>	numeric	<b>float</b>	
<b>int</b>	integer	<b>int</b>	
<b>money</b>	numeric	<b>float</b>	
<b>numeric(p,s)</b>	numeric	<b>float</b>	
<b>real</b>	numeric	<b>float</b>	
<b>smalldatetime</b>	POSIXct	<b>datetime</b>	Represented as GMT
<b>smallint</b>	integer	<b>int</b>	
<b>smallmoney</b>	numeric	<b>float</b>	
<b>tinyint</b>	integer	<b>int</b>	
<b>uniqueidentifier</b>	character	<b>varchar(max)</b>	
<b>varbinary(n)</b> n <= 8000	raw	<b>varbinary(max)</b>	Only allowed as input parameter and output
<b>varbinary(max)</b>	raw	<b>varbinary(max)</b>	Only allowed as input parameter and output
<b>varchar(n)</b> n <= 8000	character	<b>varchar(max)</b>	

## Data types not supported by R

Of the categories of data types supported by the [SQL Server type system](#), the following types are likely to pose problems when passed to R code:

- Data types listed in the **Other** section of the SQL type system topic: **cursor, timestamp, hierarchyid, uniqueidentifier, sql\_variant, xml, table**
- All spatial types
- **image**

## Data types that might convert poorly

- Most datetime types should work, except for **datetimeoffset**
- Most numeric data types are supported, but conversions might fail for **money** and **smallmoney**
- **varchar** is supported, but because SQL Server uses Unicode as a rule, use of **nvarchar** and other Unicode text data types is recommended where possible.
- Functions from the RevoScaleR library prefixed with rx can handle the SQL binary data types (**binary** and **varbinary**), but in most scenarios special handling will be required for these types. Most R code cannot work with binary columns.

For more information about SQL Server data types, see [Data Types \(Transact-SQL\)](#)

## Changes in data types between SQL Server 2016 and earlier versions

Microsoft SQL Server 2016 and Microsoft Azure SQL Database include improvements in data type conversions and in several other operations. Most of these improvements offer increased precision when you deal with floating-point types, as well as minor changes to operations on classic **datetime** types.

These improvements are all available by default when you use a database compatibility level of 130 or later. However, if you use a different compatibility level, or connect to a database using an older version, you might see differences in the precision of numbers or other results.

For more information, see [SQL Server 2016 improvements in handling some data types and uncommon operations](#).

## Verify R and SQL data schemas in advance

In general, whenever you have any doubt about how a particular data type or data structure is being used in R, use the `str()` function to get the internal structure and type of the R object. The result of the function is printed to the R console and is also available in the query results, in the **Messages** tab in Management Studio.

When retrieving data from a database for use in R code, you should always eliminate columns that cannot be used in R, as well as columns that are not useful for analysis, such as GUIDS (uniqueidentifier), timestamps and other columns used for auditing, or lineage information created by ETL processes.

Note that inclusion of unnecessary columns can greatly reduce the performance of R code, especially if high cardinality columns are used as factors. Therefore, we recommend that you use SQL Server system stored procedures and information views to get the data types for a given table in advance, and eliminate or convert incompatible columns. For more information, see [Information Schema Views in Transact-SQL](#)

If a particular SQL Server data type is not supported by R, but you need to use the columns of data in the R script, we recommend that you use the [CAST and CONVERT \(Transact-SQL\)](#) functions to ensure that the data type conversions are performed as intended before using the data in your R script.

### WARNING

If you use the **rxDataStep** to drop incompatible columns while moving data, be aware that the arguments `varsToKeep` and `varsToDelete` are not supported for the **RxSqlServerData** data source type.

# Examples

## Example 1: Implicit conversion

The following example demonstrates how data is transformed when making the round-trip between SQL Server and R.

The query gets a series of values from a SQL Server table, and uses the stored procedure `sp_execute_external_script` to output the values using the R runtime.

```
CREATE TABLE MyTable (
 c1 int,
 c2 varchar(10),
 c3 uniqueidentifier
);
go
INSERT MyTable VALUES(1, 'Hello', newid());
INSERT MyTable VALUES(-11, 'world', newid());
SELECT * FROM MyTable;

EXECUTE sp_execute_external_script
 @language = N'R'
 , @script = N'
inputDataSet["cR"] <- c(4, 2)
str(inputDataSet)
outputDataSet <- inputDataSet
 , @input_data_1 = N'SELECT c1, c2, c3 FROM MyTable'
 , @input_data_1_name = N'inputDataSet'
 , @output_data_1_name = N'outputDataSet'
WITH RESULT SETS((C1 int, C2 varchar(max), C3 varchar(max), C4 float));
```

## Results

	C1	C2	C3	C4
1	1	Hello	6e225611-4b58-4995-a0a5-554d19012ef1	4
1	-11	world	6732ea46-2d5d-430b-8ao1-86e7f3351c3e	2

Note the use of the `str` function in R to get the schema of the output data. This function returns the following information:

```
'data.frame': 2 obs. of 4 variables: $ c1: int 1 -11 $ c2: Factor w/ 2 levels "Hello", "world": 1 2
$ c3: Factor w/ 2 levels "6732EA46-2D5D-430B-8A01-86E7F3351C3E", ... : 2 1 $ cR: num 4 2
```

From this, you can see that the following data type conversions were implicitly performed as part of this query:

- **Column C1.** The column is represented as `int` in SQL Server, `integer` in R, and `int` in the output result set.

No type conversion was performed.

- **Column C2.** The column is represented as `varchar(10)` in SQL Server, `factor` in R, and `varchar(max)` in the output.

Note how the output changes; any string from R (either a factor or a regular string) will be represented as

**varchar(max)**, no matter what the length of the strings is.

- **Column C3.** The column is represented as **uniqueidentifier** in SQL Server, `character` in R, and **varchar(max)** in the output.

Note the data type conversion that happens. SQL Server supports the **uniqueidentifier** but R does not; therefore, the identifiers are represented as strings.

- **Column C4.** The column contains values generated by the R script and not present in the original data.

## Example 2: Dynamic column selection using R

The following example shows how you can use R code to check for invalid column types. The gets the schema of a specified table using the SQL Server system views, and removes any columns that have a specified invalid type.

```
connStr <- "Server=.;Database=TestDB;Trusted_Connection=Yes"
data <- RxSqlServerData(connectionString = connStr, sqlQuery = "SELECT COLUMN_NAME FROM
TestDB.INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = N'testdata' AND DATA_TYPE <> 'image';")
columns <- rxImport(data)
columnList <- do.call(paste, c(as.list(columns$COLUMN_NAME), sep = ","))
sqlQuery <- paste("SELECT", columnList, "FROM testdata")
```

## See Also

[Python Libraries and Data Types](#)

# Python libraries and data types

4/11/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes the Python libraries that are included with SQL Server Machine Learning Services (In-Database) and (Standalone) installations.

This article also lists unsupported data types, and lists the data type conversions that might be performed implicitly when data is passed between Python and SQL Server.

## Python Version

SQL Server 2017 CTP 2.0 includes a portion of the Anaconda distribution and Python 3.6.

A subset of the RevoScaleR functionality (rxLinMod, rxLogit, rxPredict, rxDTrees, rxBTrees, maybe a few others) is provided using Python APIs, using a new Python package **revoscalepy**. You can use this package to work with data using Pandas data frames, XDF files, or SQL data queries.

For more information, see [What Is revoscalepy?](#).

## Python and SQL Data Types

Python supports a limited number of data types in comparison to SQL Server.

As a result, whenever you use data from SQL Server in Python scripts, data might be implicitly converted to a compatible data type. However, often an exact conversion cannot be performed automatically, and an error is returned.

This table lists the implicit conversions that are provided. Other data types are not supported.

SQLTYPE	PYTHON TYPE
<b>bigint</b>	<code>numeric</code>
<b>binary</b>	<code>raw</code>
<b>bit</b>	<code>bool</code>
<b>char</b>	<code>str</code>
<b>float</b>	<code>float64</code>
<b>int</b>	<code>int32</code>
<b>nchar</b>	<code>str</code>
<b>nvarchar</b>	<code>str</code>
<b>nvarchar(max)</b>	<code>str</code>

SQLTYPE	PYTHON TYPE
<b>real</b>	<code>float32</code>
<b>smallint</b>	<code>int16</code>
<b>tinyint</b>	<code>uint8</code>
<b>varbinary</b>	<code>bytes</code>
<b>varbinary(max)</b>	<code>bytes</code>
<b>varchar(n)</b>	<code>str</code>
<b>varchar(max)</b>	<code>str</code>

# Native scoring

4/11/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic describes features in SQL Server 2017 that provide scoring on machine learning models in near realtime.

- What is native scoring vs. realtime scoring
- How it works
- Supported platforms and requirements

## What is native scoring and how is it different from realtime scoring?

In SQL Server 2016, Microsoft created an extensibility framework that allows R scripts to be executed from T-SQL. This framework supports any operation you might perform in R, ranging from simple functions to training complex machine learning models. However, the dual-process architecture requires invoking an external R process for every call, regardless of the complexity of the operation. If you are loading a pre-trained model from a table and scoring against it on data already in SQL Server, the overhead of calling the external R process represents an unnecessary performance cost.

*Scoring* is a two-step process. First, you specify a pre-trained model to load from a table. Second, pass new input data to the function, to generate prediction values (or *scores*). The input can be either tabular or single rows. You can choose to output a single column value representing a probability, or you might output several values, such as a confidence interval, error, or other useful complement to the prediction.

When the input includes many rows of data, it is usually faster to insert the prediction values into a table as part of the scoring process. Generating a single score is more typical in a scenario where you get input values from a form or user request, and return the score to a client application. To improve performance when generating successive scores, SQL Server might cache the model so that it can be reloaded into memory.

To support fast scoring, SQL Server Machine Learning Services (and Microsoft Machine Learning Server) provide built-in scoring libraries that work in R or in T-SQL. There are different options depending on which version you have.

### Native scoring

- The PREDICT function in Transact-SQL supports *native scoring* in any instance of SQL Server 2017. It requires only that you have a model already trained, which you can call using T-SQL. Native scoring using T-SQL has these advantages:
  - No additional configuration is required.
  - The R runtime is not called. There is no need to install R.

### Realtime scoring

- **sp\_rxPredict** is a stored procedure for realtime scoring that can be used to generates scores from any supported model type, without calling the R runtime.

This stored procedure is also available in SQL Server 2016, if you upgrade the R components using the standalone installer of Microsoft R Server. sp\_rxPredict is also supported in SQL Server 2017. Therefore, you might use this function when generating scores with a model type not supported by the PREDICT

function.

- The rxPredict function can be used for fast scoring within R code.

For all of these scoring methods, you must use a model that was trained using one of the supported RevoScaleR or MicrosoftML algorithms.

For an example of realtime scoring in action, see [End to End Loan ChargeOff Prediction Built Using Azure HDInsight Spark Clusters and SQL Server 2016 R Service](#)

## How native scoring works

Native scoring uses native C++ libraries from Microsoft that can read the model from a special binary format and generate scores. Because a model can be published and used for scoring without having to call the R interpreter, the overhead of multiple process interactions is reduced. Hence, native scoring supports much faster prediction performance in enterprise production scenarios.

To generate scores using this library, you call the scoring function, and pass the following required inputs:

- A compatible model. See the [Requirements](#) section for details.
- Input data, typically defined as a SQL query

The function returns predictions for the input data, together with any columns of source data that you want to pass through.

For code samples, along with instructions on how to prepare the models in the required binary format, see this article:

- [How to perform realtime scoring](#)

For a complete solution that includes native scoring, see these samples from the SQL Server development team:

- Deploy your ML script: [Using a Python model](#)
- Deploy your ML script: [Using an R model](#)

## Requirements

Supported platforms are as follows:

- SQL Server 2017 Machine Learning Services (includes Microsoft R Server 9.1.0)

Native scoring using PREDICT requires SQL Server 2017. It works on any version of SQL Server 2017, including Linux.

You can also perform realtime scoring using sp\_rxPredict. To use this stored procedure requires that you enable [SQL Server CLR integration](#).

- SQL Server 2016

Realtime scoring using sp\_rxPredict is possible with SQL Server 2016, and can also be run on Microsoft R Server. This option requires SQLCLR to be enabled, and that you install the Microsoft R Server upgrade.

For more information, see [Realtime scoring](#)

## Model preparation

- The model must be trained in advance using one of the supported **rx** algorithms. For details, see [Supported algorithms](#).
- The model must be saved using the new serialization function provided in Microsoft R Server 9.1.0. The serialization function is optimized to support fast scoring.

## Algorithms that support native scoring

- RevoScaleR models
  - `rxLinMod`
  - `rxLogit`
  - `rxBTrees`
  - `rxDtree`
  - `rxDForest`

If you need to use models from MicrosoftML, use realtime scoring with `sp_rxPredict`.

## Restrictions

The following model types are not supported:

- Models containing other, unsupported types of R transformations
- Models using the `rxGlm` or `rxNaiveBayes` algorithms in RevoScaleR
- PMML models
- Models created using other R libraries from CRAN or other repositories
- Models containing any other R transformation

# Realtime scoring

4/11/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic describes a feature available in SQL Server 2016 and SQL Server 2017 that supports scoring on machine learning models in near realtime.

## TIP

Native scoring is a special implementation of realtime scoring that uses the native T-SQL PREDICT function for very fast scoring, and is available only in SQL Server 2017. For more information, see [Native scoring](#).

## How realtime scoring works

Realtime scoring is supported in both SQL Server 2017 and SQL Server 2016, on specific model types created by using supported RevoScaleR or MicrosoftML algorithms. It uses native C++ libraries to generate scores, based on user input provided to a machine learning model stored in a special binary format.

Because a trained model can be used for scoring without having to call an external language runtime, the overhead of multiple processes is reduced. This supports much faster prediction performance for production scoring scenarios. Because the data never leaves SQL Server, results can be generated and inserted into a new table without any data translation between R and SQL.

Realtime scoring is a multi-step process:

1. The stored procedure that does scoring must be enabled on a per-database basis.
2. You load the pre-trained model in binary format.
3. You provide new input data, either tabular or single rows, as input to the model.
4. To generate scores, call the sp\_rxPredict stored procedure.

## Get started

For code examples and instructions, see [How to perform native scoring or realtime scoring](#).

For an example of how rxPredict can be used for scoring, see [End to End Loan ChargeOff Prediction Built Using Azure HDInsight Spark Clusters and SQL Server 2016 R Service](#)

## TIP

If you are working exclusively in R code, you can also use the `rxPredict` function for fast scoring.

## Requirements

Realtime scoring is supported on these platforms:

- SQL Server 2017 Machine Learning Services
- SQL Server R Services 2016, with an upgrade of the R Services instance to Microsoft R Server 9.1.0 or later
- Machine Learning Server (Standalone)

On SQL Server, you must enable the realtime scoring feature in advance. This is because the feature requires installation of CLR-based libraries into SQL Server.

For information regarding realtime scoring in a distributed environment based on Microsoft R Server, please refer to the [publishService](#) function available in the [mrsDeploy package](#), which supports publishing models for realtime scoring as a new a web service running on R Server.

## Restrictions

- The model must be trained in advance using one of the supported **rx** algorithms. For details, see [Supported algorithms](#). Realtime scoring with `sp_rxPredict` supports both RevoScaleR and MicrosoftML algorithms.
- The model must be saved using the new serialization functions: `rxSerialize` for R, and `rx_serialize_model` for Python. These serialization functions have been optimized to support fast scoring.
- Realtime scoring does not use an interpreter interpreter; therefore, any functionality that might require an interpreter is not supported during the scoring step. These might include:
  - Models using the `rxGlm` or `rxNaiveBayes` algorithms are not currently supported
  - RevoScaleR models that use an R transformation function, or a formula that contains a transformation, such as `A ~ log(B)` are not supported in realtime scoring. To use a model of this type, we recommend that you perform the transformation on the to input data before passing the data to realtime scoring.
- Realtime scoring is currently optimized for fast predictions on smaller data sets, ranging from a few rows to hundreds of thousand of rows. On very large datasets, using `rxPredict` might be faster.

## Algorithms that support realtime scoring

- RevoScaleR models

- `rxLinMod` \*
- `rxLogit` \*
- `rxBTrees` \*
- `rxDtree` \*
- `rxForest` \*

Models marked with \* also support native scoring with the PREDICT function.

- MicrosoftML models

- `rxFastTrees`
- `rxFastForest`
- `rxLogisticRegression`
- `rxOneClassSvm`
- `rxNeuralNet`
- `rxFastLinear`

- Transformations supplied by MicrosoftML

- `featurizeText`
- `concat`
- `categorical`
- `categoricalHash`
- `selectFeatures`

## Unsupported model types

The following model types are not supported:

- Models containing other, unsupported types of R transformations
- Models using the `rxGlm` or `rxNaiveBayes` algorithms in RevoScaleR
- PMML models
- Models created using other R libraries from CRAN or other repositories
- Models containing any other kind of R transformation other than those listed here

#### Known issues

- `sp_rxPredict` returns an inaccurate message when a NULL value is passed as the model:  
"System.Data.SqlTypes.SqlNullValueException:Data in Null".

## Next steps

[How to do realtime scoring](#)

# Data exploration and predictive modeling with R in SQL Server

7/11/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes improvements to the data science process that are possible through integration with SQL Server.

Applies to: SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## The Data Science Process

Data scientists often use R to explore data and build predictive models. This is typically an iterative process of trial and error until a good predictive model is reached. As an experienced data scientist, you might connect to the SQL Server database and fetch the data to your local workstation using the RODBC package, explore your data, and build a predictive model using standard R packages.

However, this approach has many drawbacks, that have hindered the wider adoption of R in the enterprise.

- Data movement can be slow, inefficient, or insecure
- R itself has performance and scale limitations

These drawbacks become more apparent when you need to move and analyze large amounts of data, or use data sets that don't fit into the memory available on your computer.

The new, scalable packages and R functions included with R Services (In-Database) help you overcome many of these challenges.

## What's Different about RevoScaleR?

The **RevoScaleR** package contains implementations of some of the most popular R functions, which have been redesigned to provide parallelism and scale. For more information, see [Distributed Computing using RevoScaleR](#).

The RevoScaleR package also provides support for changing *execution context*. What this means is that, for an entire solution or for just one function, you can indicate that computations should be performed using the resources of the computer that hosts the SQL Server instance, rather than your local workstation. There are multiple advantages to doing this: you avoid unnecessary data movement, and you can leverage greater computation resources on the server computer.

## R Environment and Packages

The R environment supported in R Services (In-Database) consists of a runtime, the open source language, and a graphical engine supported and extended by multiple packages. The language allows a variety of extensions that are implemented using packages.

### Using Other R Packages

In addition to the proprietary R libraries included with Microsoft Machine Learning, you can use almost any R packages in your solution, including:

- General purpose R packages from public repositories. You can obtain the most popular open source R

packages from public repositories such as CRAN, which hosts over 6000 packages that can be used by data scientists.

For the Windows platform, R packages are provided as zip files and can be downloaded and installed under the GPL license.

For information about how to install third-party packages for use with R Services (In-Database), see [Install Additional R Packages on SQL Server](#)

- Additional packages and libraries provided by R Services (In-Database).

The **RevoScaleR** package includes high performance big data analytics, improved versions of functions that support common data science tasks, optimized learners for Naive Bayes, linear regression, time series models, and neural networks, and advanced math libraries.

The **RevoPemaR** package lets you develop your own parallel external memory algorithms in R.

For more information about these packages and how to use them, see [What is RevoScaleR](#) and [Get started with RevoPemaR](#).

- **MicrosoftML** contains a collection of highly optimized machine learning algorithms and data transformations from the Microsoft Data Science team. Many of the algorithms are also used in Azure Machine Learning. For more information, see [Using the MicrosoftML Package](#).

## R Development Tools

When developing your R solution, be sure to download Microsoft R Client. This free download includes the libraries needed to support remote compute contexts and scalable algorithms:

- **Microsoft R Open:** A distribution of the R runtime and a set of packages, such as the Intel math kernel library, that boost the performance of standard R operations.
- **RevoScaleR:** An R package that lets you push computations to an instance of SQL Server. Microsoft R Enterprise. It also includes a set of common R functions that have been redesigned to provide better performance and scalability. You can identify these improved functions by the **rx** prefix. It also includes enhanced data providers for a variety of sources; these functions are prefixed with **Rx**.

You can use any Windows-based code editor that supports R, such as R Tools for Visual Studio or RStudio. The download of Microsoft R Open also includes common command-line tools for R such as RGui.exe.

## Use New Data Sources and Compute Contexts

When using the RevoScaleR package to connect to SQL Server, look for these functions to use in your R code:

- **RxSqlServerData** is a function provided in the RevoScaleR package to support improved data connectivity to SQL Server.

You use this function in your R code to define the *data source*. The data source object specifies the server and tables where the data resides and manages the task of reading data from and writing data to SQL Server.

- The **RxInSqlServer** function can be used to specify the *compute context*. In other words, you can indicate where the R code should be executed: on your local workstation, or on the computer that hosts the SQL Server instance. For more information, see [RevoScaleR Functions](#).

When you set the compute context, it affects only computations that support remote execution context, which means R operations provided by the RevoScaleR package and related functions. Typically, R solutions based on standard CRAN packages cannot run in a remote compute context, though they can be run on the SQL Server computer if started by T-SQL. However, you can use the **rxExec** function to call individual R

functions and run them remotely in SQL Server.

For examples of how to create and work with data sources and execution contexts, see these tutorials:

- [Data Science Deep Dive](#)
- [Data Analysis using Microsoft R](#)

## Deploy R Code to Production

An important part of data science is providing your analyses to others, or using predictive models to improve business results or processes. In R Services (In-Database), it is easy to move to production when your R script or model is ready.

For more information about how you can move your code to run in SQL Server, see [Operationalizing Your R Code](#).

Typically the deployment process begins with cleaning up your script to eliminate code that is not needed in production. As you move computations closer to the data, you might find ways to more efficiently move, summarize, or present data than doing everything in R. We recommend that the data scientist consult with a database developer about ways to improve performance, especially if the solution does data cleansing or feature engineering that might be more effective in SQL. Changes to ETL processes might be needed to ensure that workflows for building or scoring a model don't fail, and that input data is available in the right format.

## See Also

[Comparison of Base R and ScaleR Functions](#)

[ScaleR Functions for Working with SQL Server](#)

# How to perform realtime scoring or native scoring in SQL Server

7/16/2018 • 7 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides instructions and sample code for how to execute the realtime scoring and native scoring features in SQL Server 2017 and SQL Server 2016. The goal of both realtime scoring and native scoring is to improve the performance of scoring operations in small batches.

Both realtime scoring and native scoring are designed to let you use a machine learning model without having to install R. All you need to do is obtain a pretrained model in a compatible format, and save it in a SQL Server database.

## Choosing a scoring method

The following options are supported for fast batch prediction:

- **Native scoring:** T-SQL PREDICT function in SQL Server 2017
- **Realtime scoring:** Using the sp\_rxPredict stored procedure in either SQL Server 2016 or SQL Server 2017.

### NOTE

Use of the PREDICT function is recommended in SQL Server 2017. To use sp\_rxPredict requires that you enable SQLCLR integration. Consider the security implications before you enable this option.

The overall process of preparing the model and then generating scores is similar:

1. Create a model using a supported algorithm.
2. Serialize the model using a special binary format.
3. Make the model available to SQL Server. Typically this means storing the serialized model in a SQL Server table.
4. Call the function or stored procedure, and pass the model and input data.

### Requirements

- The PREDICT function is available in all editions of SQL Server 2017 and is enabled by default. You do not need to install R or enable additional features.
- If using sp\_rxPredict, some additional steps are required. See [Enable realtime scoring](#).
- At this time, only RevoScaleR and MicrosoftML can create compatible models. Additional model types might become available in future. For the list of currently supported algorithms, see [Realtime scoring](#).

### Serialization and storage

To use a model with either of the fast scoring options, save the model using a special serialized format, which has been optimized for size and scoring efficiency.

- Call `rxSerializeModel` to write a supported model to the **raw** format.
- Call `rxUnserializeModel` to reconstitute the model for use in other R code, or to view the model.

For more information, see [rxSerializeModel](#).

## Using SQL

From SQL code, you can train the model using `sp_execute_external_script`, and directly insert the trained models into a table, in a column of type **varbinary(max)**.

For a simple example, see [this tutorial](#)

## Using R

From R code, there are two ways to save the model to a table:

- Call the `rxWriteObject` function, from the RevoScaleR package, to write the model directly to the database.

The `rxWriteObject()` function can retrieve R objects from an ODBC data source like SQL Server, or write objects to SQL Server. The API is modeled after a simple key-value store.

If you use this function, be sure to serialize the model using the new serialization function first. Then, set the `serialize` argument in `rxWriteObject` to FALSE, to avoid repeating the serialization step.

- You can also save the model in raw format to a file and then read from the file into SQL Server. This option might be useful if you are moving or copying models between environments.

## Native scoring with PREDICT

In this example, you create a model, and then call the realtime prediction function from T-SQL.

### Step 1. Prepare and save the model

Run the following code to create the sample database and required tables.

```
CREATE DATABASE NativeScoringTest;
GO
USE NativeScoringTest;
GO
DROP TABLE IF EXISTS iris_rx_data;
GO
CREATE TABLE iris_rx_data (
 "Sepal.Length" float not null, "Sepal.Width" float not null
 , "Petal.Length" float not null, "Petal.Width" float not null
 , "Species" varchar(100) null
);
GO
```

Use the following statement to populate the data table with data from the **iris** dataset.

```
INSERT INTO iris_rx_data ("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width" , "Species")
EXECUTE sp_execute_external_script
 @language = N'R'
 , @script = N'iris_data <- iris;'
 , @input_data_1 = N''
 , @output_data_1_name = N'iris_data';
GO
```

Now, create a table for storing models.

```

DROP TABLE IF EXISTS ml_models;
GO
CREATE TABLE ml_models (model_name nvarchar(100) not null primary key
 , model_version nvarchar(100) not null
 , native_model_object varbinary(max) not null);
GO

```

The following code creates a model based on the **iris** dataset and saves it to the table named **models**.

```

DECLARE @model varbinary(max);
EXECUTE sp_execute_external_script
 @language = N'R'
 , @script = N'
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dtree <- rxDTree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data =
iris[iris.sub,])
model <- rxSerializeModel(iris.dtree, realtimeScoringOnly = TRUE)

, @params = N'@model varbinary(max) OUTPUT'
, @model = @model OUTPUT
INSERT [dbo].[ml_models]([model_name], [model_version], [native_model_object])
VALUES('iris.dtree','v1', @model) ;

```

#### NOTE

Be sure to use the [rxSerializeModel](#) function from RevoScaleR to save the model. The standard R `serialize` function cannot generate the required format.

You can run a statement such as the following to view the stored model in binary format:

```

SELECT *, datalength(native_model_object)/1024. as model_size_kb
FROM ml_models;

```

#### Step 2. Run PREDICT on the model

The following simple PREDICT statement gets a classification from the decision tree model using the **native scoring** function. It predicts the iris species based on attributes you provide, petal length and width.

```

DECLARE @model varbinary(max) =
 SELECT native_model_object
 FROM ml_models
 WHERE model_name = 'iris.dtree'
 AND model_version = 'v1';
SELECT d.* , p.*
 FROM PREDICT(MODEL = @model, DATA = dbo.iris_rx_data as d)
 WITH(setosa_Pred float, versicolor_Pred float, virginica_Pred float) as p;
go

```

If you get the error, "Error occurred during execution of the function PREDICT. Model is corrupt or invalid", it usually means that your query didn't return a model. Check whether you typed the model name correctly, or if the models table is empty.

#### NOTE

Because the columns and values returned by **PREDICT** can vary by model type, you must define the schema of the returned data by using a **WITH** clause.

# Realtime scoring with sp\_rxPredict

This section describes the steps required to set up **realtime** prediction, and provides an example of how to call the function from T-SQL.

## Step 1. Enable the realtime scoring procedure

You must enable this feature for each database that you want to use for scoring. The server administrator should run the command-line utility, RegisterRExt.exe, which is included with the RevoScaleR package.

### NOTE

In order for realtime scoring to work, SQL CLR functionality needs to be enabled in the instance; additionally, the database needs to be marked trustworthy. When you run the script, these actions are performed for you. However, consider the additional security implications before doing this!

1. Open an elevated command prompt, and navigate to the folder where RegisterRExt.exe is located. The following path can be used in a default installation:

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\
```

2. Run the following command, substituting the name of your instance and the target database where you want to enable the extended stored procedures:

```
RegisterRExt.exe /installRts [/instance:name] /database:databaseName
```

For example, to add the extended stored procedure to the CLRPredict database on the default instance, type:

```
RegisterRExt.exe /installRts /database:CLRPredict
```

The instance name is optional if the database is on the default instance. If you are using a named instance, you must specify the instance name.

3. RegisterRExt.exe creates the following objects:

- Trusted assemblies
- The stored procedure `sp_rxPredict`
- A new database role, `rxpredict_users`. The database administrator can use this role to grant permission to users who use the realtime scoring functionality.

4. Add any users who need to run `sp_rxPredict` to the new role.

### NOTE

In SQL Server 2017, additional security measures are in place to prevent problems with CLR integration. These measures impose additional restrictions on the use of this stored procedure as well.

## Step 2. Prepare and save the model

The binary format required by `sp_rxPredict` is the same as the format required to use the PREDICT function. Therefore, in your R code, include a call to `rxSerializeModel`, and be sure to specify `realtimeScoringOnly = TRUE`, as in this example:

```
model <- rxSerializeModel(model.name, realtimeScoringOnly = TRUE)
```

## Step 3. Call `sp_rxPredict`

You call `sp_rxPredict` as you would any other stored procedure. In the current release, the stored procedure takes only two parameters: `@model` for the model in binary format, and `@inputData` for the data to use in scoring, defined as a valid SQL query.

Because the binary format is the same that is used by the `PREDICT` function, you can use the models and data table from the preceding example.

```
DECLARE @irismodel varbinary(max)
SELECT @irismodel = [native_model_object] from [ml_models]
WHERE model_name = 'iris.dtreet'
AND model_version = 'v1'

EXEC sp_rxPredict
@model = @irismodel,
@inputData = N'SELECT * FROM iris_rx_data'
```

#### NOTE

The call to `sp_rxPredict` fails if the input data for scoring does not include columns that match the requirements of the model. Currently, only the following .NET data types are supported: double, float, short, ushort, long, ulong and string.

Therefore, you might need to filter out unsupported types in your input data before using it for realtime scoring.

For information about corresponding SQL types, see [SQL-CLR Type Mapping](#) or [Mapping CLR Parameter Data](#).

## Disable realtime scoring

To disable realtime scoring functionality, open an elevated command prompt, and run the following command:

```
RegisterRExt.exe /uninstallrts /database:<database_name> [/instance:name]
```

## Realtime scoring in Microsoft R Server or Machine Learning Server

Machine Learning Server supports distributed realtime scoring from models published as a web service. For more information, see these articles:

- [What are web services in Machine Learning Server?](#)
- [What is operationalization?](#)
- [Deploy a Python model as a web service with azureml-model-management-sdk](#)
- [Publish an R code block or a realtime model as a new web service](#)
- [mrsdeploy package for R](#)

# Save and load R objects from SQL Server using ODBC

4/12/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server R Services can store serialized R objects in a table and then load the object from the table as needed, without you having to re-run the R code or retrain the model. This ability to save R objects in a database is critical for scenarios such as training and saving a model, and then using it later for scoring or analysis.

To improve performance of this critical step, the **RevoScaleR** package now includes new serialization and deserialization functions that greatly improve performance, and store the object more compactly. This article describes these functions and how to use them.

## Overview

The **RevoScaleR** package now includes new functions that make it easier to save R objects to SQL Server and then read the objects from the SQL Server table. In general, each function call uses a simple key value store, in which the key is the name of the object, and the value associated with the key is the varbinary R object to be moved in or out of a table.

To save R objects to SQL Server directly from an R environment, you must:

- established a connection to SQL Server using the *RxDbcData* data source.
- Call the new functions over the ODBC connection
- Optionally, you can specify that the object not be serialized. Then, choose a new compression algorithm to use instead of the default compression algorithm.

By default, any object that you call from R to move to SQL Server is serialized and compressed. Conversely, when you load an object from a SQL Server table to use in your R code, the object is deserialized and decompressed.

## List of new functions

- `rxWriteObject` writes an R object into SQL Server using the ODBC data source.
- `rxReadObject` reads an R object from a SQL Server database, using an ODBC data source
- `rxDeleteObject` deletes an R object from the SQL Server database specified in the ODBC data source. If there are multiple objects identified by the key/version combination, all are deleted.
- `rxListKeys` lists as key-value pairs all the available objects. This helps you determine the names and versions of the R objects.

For detailed help on the syntax of each function, use R help. Details are also available in the [ScaleR reference](#).

## How to store R objects in SQL Server using ODBC

This procedure demonstrates how you can use the new functions to create a model and save it to SQL Server.

1. Set up the connection string for the SQL Server.

```
R conStr <- 'Driver={SQL Server};Server=localhost;Database=storedb;Trusted_Connection=true'
```

2. Create an *rxOdbcData* data source object in R using the connection string.

```
ds <- RxOdbcData(table="robjects", connectionString=conStr)
```

3. Delete the table if it already exists, and you don't want to track old versions of the objects.

```
if(rxSqlServerTableExists(ds$table, ds@connectionString)) {
 rxSqlServerDropTable(ds$table, ds@connectionString)
}
```

4. Define a table that can be used to store binary objects.

```
ddl <- paste(" CREATE TABLE [", ds$table, "]
[", " [id] varchar(200) NOT NULL,
[", " [value] varbinary(max),
[", " CONSTRAINT unique_id UNIQUE (id))",
sep = "")
```

5. Open the ODBC connection to create the table, and when the DDL statement has completed, close the connection.

```
rxOpen(ds, "w")
rxExecuteSQLDDL(ds, ddl)
rxClose(ds)
```

6. Generate the R objects that you want to store.

```
infertLogit <- rxLogit(case ~ age + parity + education + spontaneous + induced,
data = infert)
```

7. Use the *RxOdbcData* object created earlier to save the model to the database.

```
rxWriteObject(ds, "logit.model", infertLogit)
```

## How to read R objects from SQL Server using ODBC

This procedure demonstrates how you can use the new functions to load a model from SQL Server.

1. Set up the connection string for the SQL Server.

```
conStr2 <- 'Driver={SQL Server};Server=localhost;Database=storedb;Trusted_Connection=true'
```

2. Create an *rxOdbcData* data source object in R, using the connection string.

```
ds <- RxOdbcData(table="robjects", connectionString=conStr2)
```

3. Read the model from the table by specifying its R object name.

```
infertLogit2 <- rxReadObject(ds, "logit.model")
```

# Convert R code for execution in SQL Server (In-Database) instances

4/12/2018 • 8 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides high-level guidance on how to modify R code to work in SQL Server.

When you move R code from R Studio or another environment to SQL Server, most often the code works without further modification: for example, if the code is simple, such as a function that takes some inputs and returns a value. It is also easier to port solutions that use the **RevoScaleR** or **MicrosoftML** packages, which support execution in different execution contexts with minimal changes.

However, your code might require substantial changes if any of the following apply:

- You use R libraries that access the network or that cannot be installed on SQL Server.
- The code makes separate calls to data sources outside SQL Server, such as Excel worksheets, files on shares, and other databases.
- You want to run the code in the `@script` parameter of `sp_execute_external_script` and also parameterize the stored procedure.
- Your original solution includes multiple steps that might be more efficient in a production environment if executed independently, such as data preparation or feature engineering vs. model training, scoring, or reporting.
- You want to improve optimize performance by changing libraries, using parallel execution, or offloading some processing to SQL Server.

## Step 1. Plan requirements and resources

### Packages

- Determine which packages are needed and ensure that they work on SQL Server.
- Install packages in advance, in the default package library used by Machine Learning Services. User libraries are not supported.

### Data sources

- If you intend to embed your R code in `sp_execute_external_script`, identify primary and secondary data sources.
  - **Primary** data sources are large datasets, such as model training data, or input data for predictions. Plan to map your largest dataset to the input parameter of `sp_execute_external_script`.
  - **Secondary** data sources are typically smaller data sets, such as lists of factors, or additional grouping variables.

Currently, `sp_execute_external_script` supports only a single dataset as input to the stored procedure. However, you can add multiple scalar or binary inputs.

Stored procedure calls preceded by `EXECUTE` cannot be used as an input to `sp_execute_external_script`. You can use queries, views, or any other valid `SELECT` statement.

- Determine the outputs you need. If you run R code using `sp_execute_external_script`, the stored procedure can output just one data frame as a result. However, you can also output multiple scalar outputs, including plots and models in binary format, as well as other scalar values derived from R code or SQL parameters.

## Data types

- Make a checklist of possible data type issues.

All R data types are supported by SQL Server machine Learning Services. However, SQL Server supports a greater variety of data types than does R. Therefore, some implicit data type conversions are performed when sending SQL Server data to R, and vice versa. You might need to explicitly cast or convert some data.

NULL values are supported. However, R uses the `na` data construct to represent a missing value, which is similar to a null.

- Consider eliminating dependency on data that cannot be used by R: for example, rowid and GUID data types from SQL Server cannot be consumed by R and generate errors.

For more information, see [R Libraries and Data Types](#).

## Step 2. Convert or repackaged code

How much you change your code depends on whether you intend to submit the R code from a remote client to run in the SQL Server compute context, or intend to deploy the code as part of a stored procedure, which can provide better performance and data security. Wrapping your code in a stored procedure imposes some additional requirements.

- Define your primary input data as a SQL query wherever possible, to avoid data movement.
- When running R in a stored procedure, you can pass through multiple **scalar** inputs. For any parameters that you want to use in the output, add the **OUTPUT** keyword.

For example, the following scalar input `@model_name` contains the model name, which is also output in its own column in the results:

```
EXEC sp_execute_external_script @model_name="DefaultModel" OUTPUT, @language=N'R', @script=N'R code here'
```

- Any variables that you pass in as parameters of the stored procedure `sp_execute_external_script` must be mapped to variables in the R code. By default, variables are mapped by name.

All columns in the input dataset must also be mapped to variables in the R script. For example, assume your R script contains a formula like this one:

```
formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
```

An error is raised if the input dataset does not contain columns with the matching names `ArrDelay`, `CRSDepTime`, `DayOfWeek`, `CRSDepHour`, and `DayOfWeek`.

- In some cases, an output schema must be defined in advance for the results.

For example, to insert the data into a table, you must use the **WITH RESULT SET** clause to specify the schema.

The output schema is also required if the R script uses the argument `@parallel=1`. The reason is that multiple processes might be created by SQL Server to run the query in parallel, with the results collected at the end. Therefore, the output schema must be prepared before the parallel processes can be created.

In other cases, you can omit the result schema by using the option **WITH RESULT SETS UNDEFINED**. This statement returns the dataset from the R script without naming the columns or specifying the SQL data types.

- Consider generating timing or tracking data using T-SQL rather than R.

For example, you could pass the system time or other information used for auditing and storage by adding a T-SQL call that is passed through to the results, rather than generating similar data in the R script.

## Improve performance and security

- Avoid writing predictions or intermediate results to file. Write predictions to a table instead, to avoid data movement.
- Run all queries in advance, and review the SQL Server query plans to identify tasks that can be performed in parallel.

If the input query can be parallelized, set `@parallel=1` as part of your arguments to [sp\\_execute\\_external\\_script](#).

Parallel processing with this flag is typically possible any time that SQL Server can work with partitioned tables or distribute a query among multiple processes and aggregate the results at the end. Parallel processing with this flag is typically not possible if you are training models using algorithms that require all data to be read, or if you need to create aggregates.

- Review your R code to determine if there are steps that can be performed independently, or performed more efficiently, by using a separate stored procedure call. For example, you might get better performance by doing feature engineering or feature extraction separately, and saving the values to a table.
- Look for ways to use T-SQL rather than R code for set-based computations.

For example, this R solution shows how user-defined T-SQL functions and R can perform the same feature engineering task: [Data Science End-to-End Walkthrough](#).

- If possible, replace conventional R functions with **ScaleR** functions that support distributed execution. For more information, see [Comparison of Base R and Scale R Functions](#).
- Consult with a database developer to determine ways to improve performance by using SQL Server features such as [memory-optimized tables](#), or, if you have Enterprise Edition, [Resource Governor](#).

For more information, see [SQL Server Optimization Tips and Tricks for Analytics Services](#)

## Step 3. Prepare for deployment

- Notify the administrator so that packages can be installed and tested in advance of deploying your code.

In a development environment, it might be okay to install packages as part of your code, but this is a bad practice in a production environment.

User libraries are not supported, regardless of whether you are using a stored procedure or running R code in the SQL Server compute context.

## Package your R code in a stored procedure

- If your code is relatively simple, you can embed it in a T-SQL user-defined function without modification, as described in these samples:
  - [Create an R function that runs in rxExec](#)
  - [Feature engineering using T-SQL and R](#)
- If the code is more complex, use the R package **sqlrutils** to convert your code. This package is designed to help experienced R users write good stored procedure code.

The first step is to rewrite your R code as a single function with clearly defined inputs and outputs.

Then, use the **sqlrutils** package to generate the input and outputs in the correct format. The **sqlrutils** package generates the complete stored procedure code for you, and can also register the stored procedure in the database.

For more information and examples, see [SqlRUtils](#).

## Integrate with other workflows

- Leverage T-SQL tools and ETL processes. Perform feature engineering, feature extraction, and data cleansing in advance as part of data workflows.

When you are working in a dedicated R development environment such as R Tools for Visual Studio or R Studio, you might pull data to your computer, analyze the data iteratively, and then write out or display the results.

However, when standalone R code is migrated to SQL Server, much of this process can be simplified or delegated to other SQL Server tools.

- Use secure, asynchronous visualization strategies.

Users of SQL Server often cannot access files on the server, and SQL client tools typically do not support the R graphics device. If you generate plots or other graphics as part of the solution, consider exporting the plots as binary data and saving to a table, or writing.

- Wrap prediction and scoring functions in stored procedures for direct access by applications.

## Other resources

To view examples of how an R solution can be deployed in SQL Server, see these samples:

- [Build a predictive model for ski rental business using R and SQL Server](#)
- [In-Database Analytics for the SQL Developer](#) Demonstrates how you can make your R code more modular by wrapping it in stored procedures
- [End-to-End Data Science Solution](#) Includes a comparison of feature engineering in R and T-SQL

# Creating multiple models using rxExecBy

4/12/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server 2017 CTP 2.0 includes a new function, **rxExecBy**, that supports parallel processing of multiple related models. Rather than train one very large model based on data from multiple similar entities, the data scientist can very quickly create many related models, each using data specific to a single entity.

For example, suppose you are monitoring device failures, and capturing data for many different types of equipment. By using rxExecBy, you can provide a single large dataset as input, specify a column on which to stratify the dataset, such as device type, and then create multiple models for individual devices.

This process has been termed "pleasingly parallel" processing, because it takes a task that was somewhat onerous for the data scientist, or at best tedious, and makes it a fast, simple operation.

Typical applications of this approach include forecasting for individual household smart meters, creating revenue projections for separate product lines, or creating models for loan approvals that are tailored to individual bank branches.

## How rxExec Works

The rxExecBy function in RevoScaleR is designed for high-volume parallel processing over a large number of small data sets.

1. You call the rxExecBy function as part of your R code, and pass a dataset of unordered data.
2. Specify the partition by which the data should be grouped and sorted.
3. Define a transformation or modeling function that should be applied to each data partition
4. When the function executes, the data queries are processed in parallel if your environment supports it.  
Moreover, the modeling or transformation tasks are distributed among individual cores and executed in parallel.  
Supported compute context for these operations include RxSpark and RxInSqlServer.
5. Multiple results are returned.

## rxExecBy Syntax and Examples

**rxExecBy** takes four inputs, one of the inputs being a dataset or data source object that can be partitioned on a specified **key** column. The function returns an output for each partition. The form of the output depends on the function that is passed as an argument. For example, if you pass a modeling function such as rxLinMod, you could return a separate trained model for each partition of the dataset.

### Supported functions

Modeling: `rxLinMod`, `rxLogit`, `rxGlm`, `rxDtree`

Scoring: `rxPredict`,

Transformation or analysis: `rxCovCor`

## Example

The following example demonstrates how to create multiple models using the Airline dataset, which is partitioned on the [DayOfWeek] column. The user-defined function, `delayFunc`, is applied to each of the partitions by calling

`rxExecBy`. The function creates separate models for Mondays, Tuesdays, and so forth.

```
EXEC sp_execute_external_script
@language = N'R'
, @script = N'
delayFunc <- function(key, data, params) {
 df <- rxImport(inData = airlineData)
 rxLinMod(ArrDelay ~ CRSDepTime, data = df)
}
OutputDataSet <- rxExecBy(airlineData, c("DayOfWeek"), delayFunc)
'
, @input_data_1 = N'select ArrDelay, DayOfWeek, CRSDepTime from AirlineDemoSmall1'
, @input_data_1_name = N'airlineData'
```

If you get the error, `varsToPartition is invalid`, check whether the name of the key column or columns is typed correctly. The R language is case-sensitive.

Note that this example is not optimized for SQL Server, and you could in many cases achieve better performance by using SQL to group the data. However, using `rxExecBy`, you can create parallel jobs from R.

The following example illustrates the process in R, using SQL Server as the compute context:

```
sqlServerConnString <- "SERVER=hostname;DATABASE=TestDB;UID=DBUser;PWD=Password;"
inTable <- paste("airlinedemosmall")
sqlServerDataDS <- RxSqlServerData(table = inTable, connectionString = sqlServerConnString)

user function
".Count" <- function(keys, data, params)
{
 myDF <- rxImport(inData = data)
 return (nrow(myDF))
}

Set SQL Server compute context with level of parallelism = 2
sqlServerCC <- RxInSqlServer(connectionString = sqlServerConnString, numTasks = 4)
rxSetComputeContext(sqlServerCC)

Execute rxExecBy in SQL Server compute context
sqlServerCCResults <- rxExecBy(inData = sqlServerDataDS, keys = c("DayOfWeek"), func = .Count)
```

# Using data from OLAP cubes in R

4/12/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The **olapR** package is an R package, provided by Microsoft for use with Machine Learning Server and SQL Server, that lets you run MDX queries to get data from OLAP cubes. With this package, you don't need to create linked servers or clean up flattened rowsets; you can get OLAP data directly from R.

This article describes the API, along with an overview of OLAP and MDX for R users who might be new to multidimensional cube databases.

## IMPORTANT

An instance of Analysis Services can support either conventional multidimensional cubes, or tabular models, but an instance cannot support both types of models. Therefore, before you try to build an MDX query against a cube, verify that the Analysis Services instance contains multidimensional models.

## What is an OLAP cube?

OLAP is short for Online Analytical Processing. OLAP solutions are widely used for capturing and storing critical business data over time. OLAP data is consumed for business analytics by a variety of tools, dashboards, and visualizations. For more information, see [Online analytical processing](#).

Microsoft provides [Analysis Services](#), which lets you design, deploy, and query OLAP data in the form of *cubes* or *tabular models*. A cube is a multi-dimensional database. *Dimensions* are like facets of the data, or factors in R: you use dimensions to identify some particular subset of data that you want to summarize or analyze. For example, time is an important dimension, so much so that many OLAP solutions include multiple calendars defined by default, to use when slicing and summarizing data.

For performance reasons, an OLAP database often calculates summaries (or *aggregations*) in advance, and then stores them for faster retrieval. Summaries are based on *measures*, which represent formulas that can be applied to numerical data. You use the dimensions to define a subset of data, and then compute the measure over that data. For example, you would use a measure to compute the total sales for a certain product line over multiple quarters minus taxes, to report the average shipping costs for a particular supplier, year-to-date cumulative wages paid, and so forth.

MDX, short for multidimensional expressions, is the language used for querying cubes. An MDX query typically contains a data definition that includes one or more dimensions, and at least one measure, though MDX queries can get considerably more complex, and include rolling windows, cumulative averages, sums, ranks, or percentiles.

Here are some other terms that might be helpful when you start building MDX queries:

- *Slicing* takes a subset of the cube by using values from a single dimension.
- *Dicing* creates a subcube by specifying a range of values on multiple dimensions.
- *Drill-down* navigates from a summary to details.
- *Drill-up* moves from details to a higher level of aggregation.
- *Roll-up* summarizes the data on a dimension.

- *Pivot* rotate the cube or the data selection.

## How to use olapR to create MDX queries

The following article provides detailed examples of the syntax for creating or executing queries against a cube:

- [How to create MDX queries using R](#)

## olapR API

The **olapR** package supports two methods of creating MDX queries:

- **Use the MDX builder.** Use the R functions in the package to generate a simple MDX query, by choosing a cube, and then setting axes and slicers. This is an easy way to build a valid MDX query if you do not have access to traditional OLAP tools, or don't have deep knowledge of the MDX language.

Not all MDX queries can be created by using this method, because MDX can be complex. However, this API supports most of the most common and useful operations, including slice, dice, drilldown, rollup, and pivot in N dimensions.

- **Copy-paste well-formed MDX.** Manually create and then paste in any MDX query. This option is the best if you have existing MDX queries that you want to reuse, or if the query you want to build is too complex for **olapR** to handle.

After building your MDX using any client utility, such as SSMS or Excel, save the query string. Provide this MDX string as an argument to the *SSAS query handler* in the **olapR** package. The provider sends the query to the specified Analysis Services server, and passes back the results to R.

For examples of how to build an MDX query or run an existing MDX query, see [How to create MDX queries using R](#).

## Known issues

This section lists some known issues and common questions about the **olapR** package.

### Tabular model support

If you connect to an instance of Analysis Services that contains a tabular model, the `explore` function reports success with a return value of TRUE. However, tabular model objects are different from multidimensional objects, and the structure of a multidimensional database is different from that of a tabular model.

Although DAX (Data analysis Expressions) is the language typically used with tabular models, you can design valid MDX queries against a tabular model, if you are already familiar with MDX. You cannot use the olapR constructors to build valid MDX queries against a tabular model.

However, MDX queries are an inefficient way to retrieve data from a tabular model. If you need to get data from a tabular model for use in R, consider these methods instead:

- Enable DirectQuery on the model and add the server as a linked server in SQL Server.
- If the tabular model was built on a relational data mart, obtain the data directly from the source.

### How to determine whether an instance contains tabular or multidimensional models

A single Analysis Services instance can contain only one type of model, though it can contain multiple models. The reason is that there are fundamental differences between tabular models and multidimensional models that control the way data is stored and processed. For example, tabular models are stored in memory and leverage columnstore indexes to perform very fast calculations. In multidimensional models, data is stored on disk and aggregations are defined in advance and retrieved by using MDX queries.

If you connect to Analysis Services using a client such as SQL Server Management Studio, you can tell at a glance which model type is supported, by looking at the icon for the database.

You can also view and query the server properties to determine which type of model the instance supports. The **Server mode** property supports two values: *multidimensional* or *tabular*.

See the following article for general information about the two types of models:

- [Comparing multidimensional and tabular models](#)

See the following article for information about querying server properties:

- [OLE DB for OLAP Schema Rowsets](#)

### **Writeback is not supported**

It is not possible to write the results of custom R calculations back to the cube.

In general, even when a cube is enabled for writeback, only limited operations are supported, and additional configuration might be required. We recommend that you use MDX for such operations.

- [Write-enabled dimensions](#)
- [Write-enabled partitions](#)
- [Set custom access to cell data](#)

### **Long-running MDX queries block cube processing**

Although the **olapR** package performs only read operations, long-running MDX queries can create locks that prevent the cube from being processed. Always test your MDX queries in advance so that you know how much data should be returned.

If you try to connect to a cube that is locked, you might get an error that the SQL Server data warehouse cannot be reached. Suggested resolutions include enabling remote connections, checking the server or instance name, and so forth; however, consider the possibility of a prior open connection.

An SSAS administrator can prevent locking issues by identifying and terminating open sessions. A timeout property can also be applied to MDX queries at the server level to force termination of all long-running queries.

## Resources

If you are new to OLAP or to MDX queries, see these Wikipedia articles:

- [OLAP cubes](#)
- [MDX queries](#)

# Create a stored pProcedure using sqlrutils

4/12/2018 • 7 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes the steps for converting your R code to run as a T-SQL stored procedure. For best possible results, your code might need to be modified somewhat, to ensure that all inputs can be parameterized.

## Step 1. Rewrite R Script

For the best results, you should rewrite your R code to encapsulate it as a single function.

All variables used by the function should be defined inside the function, or should be defined as input parameters. See the [sample code](#) in this article.

Also, because the input parameters for the R function will become the input parameters of the SQL stored procedure, you must ensure that your inputs and outputs conform to the following type requirements:

### Inputs

Among the input parameters there can be at most one data frame.

The objects inside the data frame, as well as all other input parameters of the function, must be of the following R data types:

- POSIXct
- numeric
- character
- integer
- logical
- raw

If an input type is not one of the above types, it needs to be serialized and passed into the function as *raw*. In this case, the function must also include code to deserialize the input.

### Outputs

The function can output one of the following:

- A data frame containing the supported data types. All objects in the data frame must use one of the supported data types.
- A named list, containing at most one data frame. All members of the list should use one of the supported data types.
- A NULL, if your function does not return any result

## Step 2. Generate Required Objects

After your R code has been cleaned up and can be called as a single function, you will use the functions in the **sqlrutils** package to prepare the inputs and outputs in a form that can be passed to the constructor that actually builds the stored procedure.

**sqlrutils** provides functions that define the input data schema and type, and define the output data schema and type. It also includes functions that can convert R objects to the required output type. You might make multiple

function calls to create the required objects, depending on the data types your code uses.

## Inputs

If your function takes inputs, for each input, call the following functions:

- `setInputData` if the input is a data frame
- `setInputParameter` for all other input types

When you make each function call, an R object is created that you will later pass as an argument to `.StoredProcedure`, to create the complete stored procedure.

## Outputs

**sqlrutils** provides multiple functions for converting R objects such as lists to the `data.frame` required by SQL Server. If your function outputs a data frame directly, without first wrapping it into a list, you can skip this step. You can also skip the conversion this step if your function returns `NULL`.

When converting a list or getting a particular item from a list, choose from these functions:

- `setOutputData` if the variable to get from the list is a data frame
- `setOutputParameter` for all other members of the list

When you make each function call, an R object is created that you will later pass as an argument to `.StoredProcedure`, to create the complete stored procedure.

## Step 3. Generate the Stored Procedure

When all input and output parameters are ready, make a call to the `.StoredProcedure` constructor.

### Usage

```
.StoredProcedure (func, spName, ..., filePath = NULL ,dbName = NULL, connectionString = NULL, batchSeparator = "GO")
```

To illustrate, assume that you want to create a stored procedure named **sp\_rsamp** with these parameters:

- Uses an existing function **foosql**. The function was based on existing code in R function **foo**, but you rewrote the function to conform to the requirements as described in [this section](#), and named the updated function as **foosql**.
- Uses the data frame **queryinput** as input
- Generates as output a data frame with the R variable name, **sqloutput**
- You want to create the T-SQL code as a file in the `C:\Temp` folder, so that you can run it using SQL Server Management Studio later

```
.StoredProcedure (foosql, sp_rsamp, queryinput, sqloutput, filePath = "C:\\\\Temp")
```

### NOTE

Because you are writing the file to the file system, you can omit the arguments that define the database connection.

The output of the function is a T-SQL stored procedure that can be executed on an instance of SQL Server 2016 (requires R Services) or SQL Server 2017 (requires Machine Learning Services with R).

For additional examples, see the package help, by calling `help(.StoredProcedure)` from an R environment.

## Step 4. Register and Run the Stored Procedure

There are two ways that you can run the stored procedure:

- Using T-SQL, from any client that supports connections to the SQL Server 2016 or SQL Server 2017 instance
- From an R environment

Both methods require that the stored procedure be registered in the database where you intend to use the stored procedure.

## Register the stored procedure

You can register the stored procedure using R, or you can run the CREATE PROCEDURE statement in T-SQL.

- Using T-SQL. If you are more comfortable with T-SQL, open SQL Server Management Studio (or any other client that can run SQL DDL commands) and execute the CREATE PROCEDURE statement using the code prepared by the `StoredProcedure` function.
- Using R. While you are still in your R environment, you can use the `register.StoredProcedure` function in **sqlrutils** to register the stored procedure with the database.

For example, you could register the stored procedure **sp\_rsamp**le in the instance and database defined in `sqlConnStr`, by making this R call:

```
register.StoredProcedure(sp_rsamp, sqlConnStr)
```

### IMPORTANT

Regardless of whether you use R or SQL, you must run the statement using an account that has permissions to create new database objects.

## Run using SQL

After the stored procedure has been created, open a connection to the SQL database using any client that supports T-SQL, and pass values for any parameters required by the stored procedure.

## Run using R

Some additional preparation is needed. If you want to execute the stored procedure from R code, rather than from SQL Server. For example, if the stored procedure requires input values, you must set those input parameters before the function can be executed, and then pass those objects to the stored procedure in your R code.

The overall process of calling the prepared SQL stored procedure is as follows:

1. Call `getInputParameters` to get a list of input parameter objects.
2. Define a `$query` or set a `$value` for each input parameter.
3. Use `execute.StoredProcedure` to execute the stored procedure from the R development environment, passing the list of input parameter objects that you set.

## Example

This example shows the before and after versions of an R script that gets data from a SQL Server database, performs some transformations on the data, and saves it to a different database.

This simple example is used only to demonstrate how you might rearrange your R code to make it easier to convert to a stored procedure.

### Before code preparation

```

sqlConnFrom <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=AirlineSrc;Trusted_Connection=Yes;"

sqlConnTo <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=AirlineTest;Trusted_Connection=Yes;"

sqlQueryAirline <- "SELECT TOP 10000 ArrDelay, CRSDepTime, DayOfWeek FROM [AirlineDemoSmall]"

dsSqlFrom <- RxSqlServerData(sqlQuery = sqlQueryAirline, connectionString = sqlConnFrom)

dsSqlTo <- RxSqlServerData(table = "cleanData", connectionString = sqlConnTo)

xFunc <- function(data) {
 data$CRSDepHour <- as.integer(trunc(data$CRSDepTime))
 return(data)
}

xVars <- c("CRSDepTime")

sqlCompute <- RxInSqlServer(numTasks = 4, connectionString = sqlConnTo)

rxOpen(dsSqlFrom)
rxOpen(dsSqlTo)

if (rxSqlServerTableExists("cleanData", connectionString = sqlConnTo)) {
 rxSqlServerDropTable("cleanData")}

rxDataStep(inData = dsSqlFrom,
 outFile = dsSqlTo,
 transformFunc = xFunc,
 transformVars = xVars,
 overwrite = TRUE)

```

#### **NOTE**

When you use an ODBC connection rather than invoking the *RxSqlServerData* function, you must open the connection using *rxOpen* before you can perform operations on the database.

#### **After code preparation**

In the updated version, the first line defines the function name. All other code from the original R solution becomes a part of that function.

```

myetl1function <- function() {
 sqlConnFrom <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=Airline01;Trusted_Connection=Yes;"
 sqlConnTo <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer02;Database=Airline02;Trusted_Connection=Yes;"

 sqlQueryAirline <- "SELECT TOP 10000 ArrDelay, CRSDepTime, DayOfWeek FROM [AirlineDemoSmall]"

 dsSqlFrom <- RxSqlServerData(sqlQuery = sqlQueryAirline, connectionString = sqlConnFrom)

 dsSqlTo <- RxSqlServerData(table = "cleanData", connectionString = sqlConnTo)

 xFunc <- function(data) {
 data$CRSDepHour <- as.integer(trunc(data$CRSDepTime))
 return(data)}
 }

 xVars <- c("CRSDepTime")

 sqlCompute <- RxInSqlServer(numTasks = 4, connectionString = sqlConnTo)

 if (rxSqlServerTableExists("cleanData", connectionString = sqlConnTo)) {rxSqlServerDropTable("cleanData")}

 rxDataStep(inData = dsSqlFrom,
 outFile = dsSqlTo,
 transformFunc = xFunc,
 transformVars = xVars,
 overwrite = TRUE)
 return(NULL)
}

```

#### NOTE

Although you do not need to open the ODBC connection explicitly as part of your code, an ODBC connection is still required to use **sqlrutils**.

## See Also

[Generating a Stored Procedure using sqlrutils](#)

# Performance tuning for R in SQL Server

4/12/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is the first in a series of four articles that describe performance optimization for R Services, based on two case studies:

- Performance tests conducted by the product development team to validate the performance profile of R solutions
- Performance optimization by the Microsoft data science team for a specific machine learning scenario often requested by customers.

The goal of this series is to provide guidance about the types of performance tuning techniques that are most useful for running R jobs in SQL Server.

- This (first) topic provides an overview of the architecture and some common problems when optimizing for data science tasks.
- The second article covers specific hardware and SQL Server optimizations.
- The third article covers optimizations in R code and resources for operationalization.
- The fourth article describes test methods in detail, and reports findings and conclusions.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Performance goals and targeted scenarios

The R Services feature was introduced in SQL Server 2016 to support execution of R scripts by SQL Server.

When you use this feature, the R runtime operates in a separate process from the database engine, but exchanges data securely with the database engine, using server resources and services that interact with SQL Server, such as the Trusted Launchpad.

In SQL Server 2017, support was announced for running Python scripts using the same architecture, with additional language to follow in future.

As the number of supported versions and language grows, it is important that the database administrator and database architect are aware of the potential for resource contention due to machine learning jobs, and that they are able to configure the server to support the new workloads. Although keeping analytics close to the data eliminates insecure data movements, it also moves analytical workloads off the laptop of the data scientist and back onto the server hosting the data.

Performance optimization for machine learning is clearly not a one-size-fits-all proposition. The following common tasks might have very different performance profiles:

- Training tasks: creating and training a regression model vs. training a neural network
- Feature engineering using R vs. feature extraction using T-SQL
- Scoring on single rows or small batches, vs. bulk scoring using tabular inputs
- Running scoring in R vs. deploying models to production on SQL Server in stored procedures
- Modifying R code to minimize data transfer or remove costly data transformations
- Enable automated testing and retraining of models

Because the choice of optimization techniques depends on which task is critical for your application or use case,

the case studies cover both general performance tips, and guidance on how to optimize for a specific scenario, optimization for batch scoring.

- **Individual optimization options in SQL Server**

In the first performance case study, multiple tests were run on a single dataset using a single type of model. The rxLinMod algorithm in RevoScaleR was used to build a model and create scores from it, but the code as well as the underlying data tables were systematically altered to test the impact of each change.

For example, in one test run, the R code was altered so that a comparison could be made between feature engineering using a transformation function vs. pre-computing the features and then loading features from a table. In another test run, model training performance was compared between using a standard indexed table vs. data in a table with various types of compression or new index types.

- **Optimization for a specific high-volume scoring scenario**

The machine learning task in the second case study involves processing many resumes submitted for multiple positions, and finding the best candidate for each job position. The machine learning model itself is formulated as a binary classification problem: it takes a resume and job description as input, and generates the probability for each resume-job pair. Because the goal is to find the best match, a user-defined probability threshold is used to further filter and get just the good matches.

For this solution, the main objective was to achieve low latency during scoring. However, this task is computationally expensive even during the scoring process, because each new job must be matched against millions of resumes within a reasonable time frame. Moreover, the feature engineering step produces over 2000 features per resume or job, and is a significant performance bottleneck.

We suggest that you review all results from the first case study to determine which techniques are applicable to your solution, and weigh their potential impact.

Then, review the results of the scoring optimization case study to see how the author applied different techniques and optimized the server to support a particular workload.

## Performance optimization process

Configuration and tuning for performance requires creating a solid base, on which to layer optimizations designed for specific workloads:

- Choose an appropriate server to host analytics. Typically, a reporting secondary, data warehouse or other server that is already used for other reporting or analytics is preferred. However, in a hybrid transactional-analytical processing (HTAP) solution, operational data can be used as the input to R for fast scoring.
- Configure the SQL Server instance to balance database engine operations and R or Python script execution at appropriate levels. This can include changing SQL Server defaults for memory and CPU usage, NUMA and processor affinity settings, and creation of resource groups.
- Optimize the server computer to support efficient use of external scripts. This can include increasing the number of accounts used for external script execution, enabling package management, and assigning users to related security roles.
- Applying optimizations specific to data storage and transfer in SQL Server, including indexing and statistics strategies, query design and query optimization, and the design of tables that are used for machine learning. You might also analyze data sources and data transport methods, or modify ETL processes to optimize feature extraction.
- Tune the analytical model to avoid inefficiencies. The scope of the optimizations that are possible depend on the complexity of your R code and the packages and data you are using. Key tasks include eliminating costly data transformations or offloading data preparation or feature engineering tasks from R or Python to

ETL processes or SQL. You might also refactor scripts, eliminate in-line package installs, divide R code into separate procedures for training, scoring, and evaluation, or simplify code to work more efficiently as a stored procedure.

## Articles in this series

- [Performance tuning for R in SQL Server - hardware](#)

Provides guidance for configuring the hardware that SQL Server is installed on, and for configuring the SQL Server instance to better support external scripts. It is particularly useful for **database administrators**.

- [Performance tuning for R in SQL Server - code and data optimization](#)

Provides specific tips on how to optimize the external script to avoid known problems. It is most useful to **data scientists**.

[!NOTE]

While much of the information in this section applies to R in general, some information is specific to RevoScaleR analytic functions. Detailed performance guidance is not available for **revoscalepy** and other supported Python libraries.

- [Performance tuning for R in SQL Server - methods and results](#)

Summarizes what data was used the two case studies, how performance was tested, and how the optimizations affected results.

# SQL Server configuration for use with R

4/12/2018 • 14 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is the second in a series that describes performance optimization for R Services based on two case studies. This article provides guidance about the hardware and network configuration of the computer that is used to run SQL Server R Services. It also contains information about ways to configure the SQL Server instance, database, or tables used in a solution. Because use of NUMA in SQL Server blurs the line between hardware and database optimizations, a third section discusses CPU affinization and resource governance in detail.

## TIP

If you are new to SQL Server, we highly recommend that you also review the SQL Server performance tuning guide: [Monitor and tune for performance](#).

## Hardware optimization

Optimization of the server computer is important for making sure that you have the resources to run external scripts. When resources are limited, you might see symptoms such as these:

- Job execution is deferred or canceled to prioritize other database operations
- Error "quota exceeded" causing R script to terminate without completion
- Data loaded into R memory truncated, for incomplete results

## Memory

The amount of memory available on the computer can have a large impact on the performance of advanced analytic algorithms. Insufficient memory might affect the degree of parallelism when using the SQL compute context. It can also affect the chunk size (rows per read operation) that can be processed, and the number of simultaneous sessions that can be supported.

A minimum of 32 GB is highly recommended. If you have more than 32 GB available, you can configure the SQL data source to use more rows in every read operation to improve performance.

You can also manage memory used by the instance. By default, SQL Server is prioritized over external script processes when memory is allocated. In a default installation of R Services, only 20% of available memory is allocated to R.

Typically this is not enough for data science tasks, but neither do you want to starve SQL server of memory. You should experiment and fine-tune memory allocation between the database engine, related services, and external scripts, with the understanding that the optimum configuration varies case by case.

For the resume-matching model, external script use was heavy and there were no other database engine services running; therefore, resources allocated to external scripts were increased to 70%, which was the best configuration for script performance.

## Power options

On the Windows operating system, the **High performance** power option should be used. Using a different power setting results in decreased or inconsistent performance when using SQL Server.

## Disk IO

Training and prediction jobs using R Services are inherently IO bound, and depend on the speed of the disk(s) that the database is stored on. Faster drives, such as solid-state drives (SSD) may help.

Disk IO is also affected by other applications accessing the disk: for example, read operations against a database by other clients. Disk IO performance can also be affected by settings on the file system in use, such as the block size used by the file system.

If multiple drives are available, store the databases on a different drive than SQL Server so that requests for the database engine are not hitting the same disk as requests for data stored in the database.

Disk IO can also greatly impact performance when running RevoScaleR analytic functions that use multiple iterations during training. For example, `rxLogit`, `rxDTree`, `rxDForest`, and `rxBTrees` all use multiple iterations. When the data source is SQL Server, these algorithms use temporary files that are optimized to capture the data. These files are automatically cleaned up after the session completes. Having a high-performance disk for read/write operations can significantly improve the overall elapsed time for these algorithms.

#### **NOTE**

Early versions of R Services required 8.3 filename support on Windows operating systems. This restriction was lifted after Service Pack 1. However, you can use `fsutil.exe` to determine whether a drive supports 8.3 filenames, or to enable support if it does not.

### **Paging file**

The Windows operating system uses a paging file to manage crash dumps and for storing virtual memory pages. If you notice excessive paging, consider increasing the physical memory on the machine. Although having more physical memory does not eliminate paging, it does reduce the need for paging.

The speed of the disk that the page file is stored on can also affect performance. Storing the page file on an SSD, or using multiple page files across multiple SSDs, can improve performance.

For information on sizing the page file, see [How to determine the appropriate page file size for 64-bit versions of Windows](#).

## **Optimizations at instance or database level**

Optimization of the SQL Server instance is the key to efficient execution of external scripts.

#### **NOTE**

The optimal settings differ depending on the size and type of your data, the number of columns you are using for scoring or training a model.

You can review the results of specific optimizations in the final article: [Performance Tuning - case study results](#)

For sample scripts, see the separate [GitHub repository](#).

### **Table compression**

IO performance can often be improved by using either compression or a columnar data store. Generally, data is often repeated in several columns within a table, so using a columnstore takes advantage of these repetitions when compressing the data.

A columnstore might not be as efficient if there are numerous insertions into the table, but is a good choice if the data is static or only changes infrequently. If a columnar store is not appropriate, enabling compression on a row major table can be used to improve IO.

For more information, see the following documents:

- [Data compression](#)
- [Enable compression on a table or index](#)
- [Columnstore indexes guide](#)

## Memory-optimized tables

Nowadays, memory is no longer a problem for modern computers. As hardware specifications continue to improve, it is relatively easy to get RAM at good values. However, at the same time, data is being produced more quickly than ever before, and the data must be processed with low latency.

Memory-optimized tables represent one solution, in that they leverage the large memory available in advanced computers to tackle the problem of big data. Memory-optimized tables mainly reside in memory, so that data is read from and written to memory. For durability, a second copy of the table is maintained on disk and data is only read from disk during database recovery.

If you need to read from and write to tables frequently, memory-optimized tables can help with high scalability and low latency. In the resume-matching scenario, use of memory-optimized tables allowed us to read all the resume features from the database and store them in main memory, to match with new job openings. This significantly reduced disk IO.

Additional performance gains were achieved by using memory-optimized table in the process of writing predictions back to the database from multiple concurrent batches. The use of memory-optimized tables on SQL Server enabled low latency on table reads and writes.

The experience was also seamless during development. Durable memory-optimized tables were created at the same time that the database was created. Therefore, development used the same workflow regardless of where the data was stored.

## Processor

SQL Server can perform tasks in parallel by using the available cores on the machine; the more cores that are available, the better the performance. While increasing the number of cores might not help for IO bound operations, CPU bound algorithms benefit from faster CPUs with many cores.

Because the server is normally used by multiple users simultaneously, the database administrator must determine the ideal number of cores that are needed to support peak workload computations.

## Resource governance

In editions that support Resource Governor, you can use resource pools to specify that certain workloads are allocated some number of CPUs. You can also manage the amount of memory allocated to specific workloads.

Resource governance in SQL Server lets you centralize monitoring and control of the various resources used by SQL Server and by R. For example, you might allocate half the available memory for the database engine, to ensure that core services can always run in spite of transient heavier workloads.

The default value for memory consumption by external scripts is limited to 20% of the total memory available for SQL Server itself. This limit is applied by default to ensure that all tasks that rely on the database server are not severely affected by long running R jobs. However, these limits can be changed by the database administrator. In many cases, the 20% limit is not adequate to support serious machine learning workloads.

The configuration options supported are **MAX\_CPU\_PERCENT**, **MAX\_MEMORY\_PERCENT**, and **MAX\_PROCESSES**. To view the current settings, use this statement:

```
SELECT * FROM sys.resource_governor_external_resource_pools
```

- If the server is primarily used for R Services, it might be helpful to increase MAX\_CPU\_PERCENT to 40% or 60%.
- If many R sessions must use the same server at the same time, all three settings should be increased.

To change the allocated resource values, use T-SQL statements.

- This statement sets the memory usage to 40%:

```
ALTER EXTERNAL RESOURCE POOL [default] WITH (MAX_MEMORY_PERCENT = 40)
```

- This statement sets all three configurable values:

```
ALTER EXTERNAL RESOURCE POOL [default] WITH (MAX_CPU_PERCENT = 40, MAX_MEMORY_PERCENT = 50,
MAX_PROCESSES = 20)
```

- If you change a memory, CPU, or max process setting, and then want to apply the settings immediately, run this statement: `ALTER RESOURCE GOVERNOR RECONFIGURE`

## Soft-NUMA, hardware NUMA, and CPU affinity

When using SQL Server as the compute context, you can sometimes achieve better performance by tuning settings related to NUMA and processor affinity.

Systems with *hardware NUMA* have more than one system bus, each serving a small set of processors. Each CPU can access memory associated with other groups in a coherent way. Each group is called a NUMA node. If you have hardware NUMA, it may be configured to use interleaved memory instead of NUMA. In that case, Windows and therefore SQL Server will not recognize it as NUMA.

You can run the following query to find the number of memory nodes available to SQL Server:

```
SELECT DISTINCT memory_node_id
FROM sys.dm_os_memory_clerks
```

If the query returns a single memory node (node 0), either you do not have hardware NUMA, or the hardware is configured as interleaved (non-NUMA). SQL Server also ignores hardware NUMA when there are four or fewer CPUs, or if at least one node has only one CPU.

If your computer has multiple processors but does not have hardware-NUMA, you can also use [Soft-NUMA](#) to subdivide CPUs into smaller groups. In both SQL Server 2016 and SQL Server 2017, the Soft-NUMA feature is automatically enabled when starting the SQL Server service.

When Soft-NUMA is enabled, SQL Server automatically manages the nodes for you; however, to optimize for specific workloads, you can disable *soft affinity* and manually configure CPU affinity for the soft NUMA nodes. This can give you more control over which workloads are assigned to which nodes, particularly if you are using an edition of SQL Server that supports resource governance. By specifying CPU affinity and aligning resource pools with groups of CPUs, you can reduce latency, and ensure that related processes are performed within the same NUMA node.

The overall process for configuring soft-NUMA and CPU affinity to support R workloads is as follows:

1. Enable soft-NUMA, if available
2. Define processor affinity
3. Create resource pools for external processes, using [Resource Governor](#)
4. Assign the [workload groups](#) to specific affinity groups

For details, including sample code, see this tutorial: [SQL Optimization Tips and Tricks \(Ke Huang\)](#)

### Other resources:

- [Soft-NUMA in SQL Server](#)

How to map soft-NUMA nodes to CPUs

- [Automatic soft-NUMA: It just runs faster \(Bob Ward\)](#)

Describes history as well as implementation details, with performance on newer multi-core servers.

## Task-specific optimizations

This section summarizes methods adopted in these case studies, and in other tests, for optimizing specific machine learning workloads. Common workloads include model training, feature extraction and feature engineering, and various scenarios for scoring: single-row, small batch, and large batch.

### Feature engineering

One pain point with R is that it is usually processed on a single CPU. This is a major performance bottleneck for many tasks, especially feature engineering. In the resume-matching solution, the feature engineering task alone created 2,500 cross-product features that had to be combined with the original 100 features. This task would take a significant amount of time if everything was done on a single CPU.

There are multiple ways to improve the performance of feature engineering. You can either optimize your R code and keep feature extraction inside the modeling process, or move the feature engineering process into SQL.

- Using R. You define a function and then pass it as the argument to `rxTransform` during training. If the model supports parallel processing, the feature engineering task can be processed using multiple CPUs. Using this approach, the data science team observed a 16% performance improvement in terms of scoring time. However, this approach requires a model that supports parallelization and a query that can be executed using a parallel plan.
- Use R with a SQL compute context. In a multiprocessor environment with isolated resources available for execution of separate batches, you can achieve greater efficiency by isolating the SQL queries used for each batch, to extract data from tables and constrain the data on the same workload group. Methods used to isolate the batches include partitioning, and use of PowerShell to execute separate queries in parallel.
- Ad hoc parallel execution: In a SQL Server compute context, you can rely on the SQL database engine to enforce parallel execution if possible and if that option is found to be more efficient.
- Use T-SQL in a separate featurization process. Precomputing the feature data using SQL is generally faster.

### Prediction (scoring) in parallel

One of the benefits of SQL Server is its ability to handle a large volume of rows in parallel. Nowhere is this advantage so marked as in scoring. Generally the model does not need access to all the data for scoring, so you can partition the input data, with each workload group processing one task.

You can also send the input data as a single query, and SQL Server then analyzes the query. If a parallel query plan can be created for the input data, it automatically partitions data assigned to the nodes and performs required joins and aggregations in parallel as well.

If you are interested in the details of how to define a stored procedure for use in scoring, see the sample project on [GitHub](#) and look for the file "step5\_score\_for\_matching.sql". The sample script also tracks query start and end times and writes the time to the SQL console so that you can assess performance.

### Concurrent scoring using resource groups

To scale up the scoring problem, a good practice is to adopt the map-reduce approach in which millions of items are divided into multiple batches. Then, multiple scoring jobs are executed concurrently. In this framework, the batches are processed on different CPU sets, and the results are collected and written back to the database.

This is the approach used in the resume-matching scenario; however, resource governance in SQL Server is essential for implementing this approach. By setting up workload groups for external script jobs, you can route R scoring jobs to different processor groups and achieve faster throughput.

Resource governance can also help allocate divide the available resources on the server (CPU and memory) to minimize workload competition. You can set up classifier functions to distinguish between different types of R

jobs: for example, you might decide that scoring called from an application always takes priority, while retraining jobs have low priority. This resource isolation can potentially improve execution time and provide more predictable performance.

### Concurrent scoring using PowerShell

If you decide to partition the data yourself, you can use PowerShell scripts to execute multiple concurrent scoring tasks. To do this, use the `Invoke-SqlCmd` cmdlet, and initiate the scoring tasks in parallel.

In the resume-matching scenario, concurrency was designed as follows:

- 20 processors divided into four groups of five CPUs each. Each group of CPUs is located on the same NUMA node.
- Maximum number of concurrent batches was set to eight.
- Each workload group must handle two scoring tasks. As soon as one task finished reading data and starts scoring, the other task can start reading data from the database.

To see the PowerShell scripts for this scenario, open the file `experiment.ps1` in the [Github project](#).

### Storing models for prediction

When training and evaluation finishes and you have selected a best model, we recommend storing the model in the database so that it is available for predictions. Loading the pre-computed model from the database for the prediction is efficient, because SQL Server machine learning uses special serialization algorithms to store and load models when moving between R and the database.

#### TIP

In SQL Server 2017, you can use the `PREDICT` function to perform scoring even if R is not installed on the server. Limited models types are supported, from the `RevoScaleR` package.

However, depending on the algorithm you use, some models can be quite large, especially when trained on a large data set. For example, algorithms such as `lm` or `glm` generate a lot of summary data along with rules. Because there are limits on the size of a model that can be stored in a varbinary column, we recommend that you eliminate unnecessary artifacts from the model before storing the model in the database for production.

## Articles in this series

[Performance tuning for R – introduction](#)

[Performance tuning for R - SQL Server configuration](#)

[Performance tuning for R - R code and data optimization](#)

[Performance Tuning - case study results](#)

# Performance for R Services - data optimization

5/29/2018 • 10 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article is the third in a series that describes performance optimization for R Services based on two case studies. This article discusses performance optimizations for R or Python scripts that run in SQL Server. It also describes methods that you can use to update your R code, both to boost performance and to avoid known issues.

## Choosing a compute context

In SQL Server 2016 and 2017, you can use either the **local** or **SQL** compute context when running R or Python script.

When using the **local** compute context, analysis is performed on your computer and not on the server. Therefore, if you are getting data from SQL Server to use in your code, the data must be fetched over the network. The performance hit incurred for this network transfer depends on the size of the data transferred, speed of the network, and other network transfers occurring at the same time.

When using the **SQL Server compute context**, the code is executed on the server. If you are getting data from SQL Server, the data should be local to the server running the analysis, and therefore no network overhead is introduced. If you need to import data from other sources, consider arranging ETL beforehand.

When working with large data sets, you should always use the SQL compute context.

## Factors

The R language has the concept of "factors", which are special variable for categorical data. Data scientists often use factor variables in their formula, because handling categorical variables as factors ensures that the data is processed properly by machine learning functions. For more information, see [R for Dummies: Factor Variables](#).

By design, factor variables can be converted from strings to integers and back again for storage or processing. The R `data.frame` function handles all strings as factor variables, unless the argument `stringsAsFactors` is set to **False**. What this means is that strings are automatically converted to an integer for processing, and then mapped back to the original string.

If the source data for factors is stored as an integer, performance can suffer, because R converts the factor integers to strings at run time, and then performs its own internal string-to-integer conversion.

To avoid such run-time conversions, consider storing the values as integers in the SQL Server table, and using the `collInfo` argument to specify the levels for the column used as factor. Most data source objects in RevoScaleR take the parameter `collInfo`. You use this parameter to name the variables used by the data source, specify their type, and define the variables levels or transformations on the column values.

For example, the following R function call gets the integers 1, 2, and 3 from a table, but maps the values to a factor with levels "apple", "orange", and "banana".

```
c("fruit" = c(type = "factor", levels=as.character(c(1:3)), newLevels=c("apple", "orange", "banana")))
```

When the source column contains strings, it is always more efficient to specify the levels ahead of time using the

`collinfo` parameter. For example, the following R code treats the strings as factors as they are being read.

```
c("fruit" = c(type = "factor", levels= c("apple", "orange", "banana")))
```

If there is no semantic difference in the model generation, then the latter approach can lead to better performance.

## Data transformations

Data scientists often use transformation functions written in R as part of the analysis. The transformation function is applied to each row retrieved from the table. In SQL Server, such transformations are applied to all rows retrieved in a batch, which requires communication between the R interpreter and the analytics engine. To perform the transformation, the data moves from SQL to the analytics engine and then to the R interpreter process and back.

For this reason, using transformations as part of your R code can have a significant adverse effect on the performance of the algorithm, depending on the amount of data involved.

It is more efficient to have all necessary columns in the table or view before performing analysis, and avoid transformations during the computation. If it is not possible to add additional columns to existing tables, consider creating another table or view with the transformed columns and use an appropriate query to retrieve the data.

## Batch row reads

If you use a SQL Server data source (`RxSqlServerData`) in your code, we recommend that you try using the parameter `rowsPerRead` to specify batch size. This parameter defines the number of rows that are queried and then sent to the external script for processing. At run time, the algorithm sees only the specified number of rows in each batch.

The ability to control the amount of data that is processed at a time can help you solve or avoid problems. For example, if your input dataset is very wide (has many columns), or if the dataset has a few large columns (such as free text), you can reduce the batch size to avoid paging data out of memory.

By default, the value of this parameter is set to 50000, to ensure decent performance even on machines with low memory. If the server has enough available memory, increasing this value to 500,000 or even a million can yield better performance, especially for large tables.

The benefits of increasing batch size become evident on a large data set, and in a task that can run on multiple processes. However, increasing this value does not always produce the best results. We recommend that you experiment with your data and algorithm to determine the optimal value.

## Parallel processing

To improve the performance of `rx` analytic functions, you can leverage the ability of SQL Server to execute tasks in parallel using available cores on the server computer.

There are two ways to achieve parallelization with R in SQL Server:

- **Use `@parallel`.** When using the `sp_execute_external_script` stored procedure to run an R script, set the `@parallel` parameter to `1`. This is the best method if your R script does **not** use RevoScaleR functions, which have other mechanisms for processing. If your script uses RevoScaleR functions (generally prefixed with "rx"), parallel processing is performed automatically and you do not need to explicitly set `@parallel` to `1`.

If the R script can be parallelized, and if the SQL query can be parallelized, then the database engine creates multiple parallel processes. The maximum number of processes that can be created is equal to the

**max degree of parallelism** (MAXDOP) setting for the instance. All processes then run the same script, but receive only a portion of the data.

Thus, this method is not useful with scripts that must see all the data, such as when training a model. However, it is useful when performing tasks such as batch prediction in parallel. For more information on using parallelism with `sp_execute_external_script`, see the **Advanced tips: parallel processing** section of [Using R Code in Transact-SQL](#).

- **Use numTasks =1.** When using `rx` functions in a SQL Server compute context, set the value of the `numTasks` parameter to the number of processes that you would like to create. The number of processes created can never be more than **MAXDOP**; however, the actual number of processes created is determined by the database engine and may be less than you requested.

If the R script can be parallelized, and if the SQL query can be parallelized, then SQL Server creates multiple parallel processes when running the `rx` functions. The actual number of processes that are created depends on a variety of factors such as resource governance, current usage of resources, other sessions, and the query execution plan for the query used with the R script.

## Query parallelization

In Microsoft R, you can work with SQL Server data sources by defining your data as an `RxSqlServerData` data source object.

Creates a data source based on an entire table or view:

```
RxSqlServerData(table= "airline", connectionString = sqlConnString)
```

Creates a data source based on a SQL query:

```
RxSqlServerData(sqlQuery= "SELECT [ArrDelay],[CRSDepTime],[DayOfWeek] FROM airlineWithIndex WHERE rowNum <= 100000", connectionString = sqlConnString)
```

### NOTE

If a table is specified in the data source instead of a query, R Services uses internal heuristics to determine the necessary columns to fetch from the table; however, this approach is unlikely to result in parallel execution.

To ensure that the data can be analyzed in parallel, the query used to retrieve the data should be framed in such a way that the database engine can create a parallel query plan. If the code or algorithm uses large volumes of data, make sure that the query given to `RxSqlServerData` is optimized for parallel execution. A query that does not result in a parallel execution plan can result in a single process for computation.

If you need to work with large datasets, use Management Studio or another SQL query analyzer before you run your R code, to analyze the execution plan. Then, take any recommended steps to improve the performance of the query. For example, a missing index on a table can affect the time taken to execute a query. For more information, see [Monitor and Tune for Performance](#).

Another common mistake that can affect performance is that a query retrieves more columns than are required. For example, if a formula is based on only three columns, but your source table has 30 columns, you are moving data unnecessarily.

- Avoid using `SELECT *!`
- Take some time to review the columns in the dataset and identify only the ones needed for analysis
- Remove from your queries any columns that contain data types that are incompatible with R code, such as

GUIDS and rowguids

- Check for unsupported date and time formats
- Rather than load a table, create a view that selects certain values or casts columns to avoid conversion errors

## Optimizing the machine learning algorithm

This section provides miscellaneous tips and resources that are specific to RevoScaleR and other options in Microsoft R.

### TIP

A general discussion of R optimization is out of the scope of this article. However, if you need to make your code faster, we recommend the popular article, [The R Inferno](#). It covers programming constructs in R and common pitfalls in vivid language and detail, and provides many specific examples of R programming techniques.

### Optimizations for RevoScaleR

Many RevoScaleR algorithms support parameters to control how the trained model is generated. While the accuracy and correctness of the model is important, the performance of the algorithm might be equally important. To get the right balance between accuracy and training time, you can modify parameters to increase the speed of computation, and in many cases, improve performance without reducing the accuracy or correctness.

- [rxDTree](#)

`rxDTree` supports the `maxDepth` parameter, which controls the depth of the decision tree. As `maxDepth` is increased, performance can degrade, so it is important to analyze the benefits of increasing the depth vs. hurting performance.

You can also control the balance between time complexity and prediction accuracy by adjusting parameters such as `maxNumBins`, `maxDepth`, `maxComplete`, and `maxSurrogate`. Increasing the depth to beyond 10 or 15 can make the computation very expensive.

- [rxLinMod](#)

Try using the `cube` argument if the first dependent variable in the formula is a factor variable.

When `cube` is set to `TRUE`, the regression is performed using a partitioned inverse, which might be faster and use less memory than standard regression computation. If the formula has a large number of variables, the performance gain can be significant.

- [rxLogit](#)

Use the `cube` argument if the first dependent variable is a factor variable.

When `cube` is set to `TRUE`, the algorithm uses a partitioned inverse, which might be faster and use less memory. If the formula has a large number of variables, the performance gain can be significant.

For additional guidance on optimization of RevoScaleR, see these articles:

- Support article: [Performance tuning options for rxDForest and rxDTree](#)
- Methods for controlling model fit in a boosted tree model: [Estimating Models Using Stochastic Gradient Boosting](#)
- Overview of how RevoScaleR moves and processes data: [Write custom chunking algorithms in ScaleR](#)
- Programming model for RevoScaleR: [Managing threads in RevoScaleR](#)
- Function reference for [rxDForest](#)

- Function reference for [rxBTrees](#)

## Use MicrosoftML

We also recommend that you look into the new **MicrosoftML** package, which provides scalable machine learning algorithms that can use the compute contexts and transformations provided by RevoScaleR.

- [Get started with MicrosoftML](#)
- [How to choose a MicrosoftML algorithm](#)

## Operationalize a solution using Microsoft R Server

If your scenario involves fast prediction using a stored model, or integrating machine learning into an application, you can use the [operationalization](#) features in Microsoft R Server (formerly known as DeployR).

- As a **data scientist**, use the [mrsdeploy package](#) to share R code with other computers, and integrate R analytics inside web, desktop, mobile, and dashboard applications: [How to publish and manage R web services in R Server](#)
- As an **administrator**, learn how to manage packages, monitor web nodes and compute nodes, and control security on R jobs: [How to interact with and consume web services in R](#)

## Articles in this series

[Performance tuning for R – introduction](#)

[Performance tuning for R - SQL Server configuration](#)

[Performance tuning for R - R code and data optimization](#)

[Performance Tuning - case study results](#)

# Performance for R Services: results and resources

4/12/2018 • 13 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This article is the fourth and final in a series that describes performance optimization for R Services. This article summarizes the methods, findings, and conclusions of two case studies that tested various optimization methods.

The two case studies had different goals:

- The first case study, by the R Services development team, sought to measure the impact of specific optimization techniques
- The second case study, by a data scientist team, experimented with multiple methods to determine the best optimizations for a specific high-volume scoring scenario.

This topic lists the detailed results of the first case study. For the second case study, a summary describes the overall findings. At the end of this topic are links to all scripts and sample data, and resources used by the original authors.

## Performance case study: Airline dataset

This case study by the SQL Server R Services development team tested the effects of various optimizations. A single rxLogit model was created and scoring performed on the Airline data set. Optimizations were applied during the training and scoring processes to assess individual impacts.

- Github: [Sample data and scripts](#) for SQL Server optimizations study

### Test methods

1. The Airline dataset consists a single table of 10M rows. It was downloaded and bulk loaded into SQL Server.
2. Six copies of the table were made.
3. Various modifications were applied to the copies of the table, to test SQL Server features such as page compression, row compression, indexing, columnar data store, etc.
4. Performance was measured before and after each optimization was applied.

TABLE NAME	DESCRIPTION
<i>airline</i>	Data converted from original xdf file using <code>rxDataStep</code>
<i>airlineWithIntCol</i>	<i>DayOfWeek</i> converted to an integer rather than a string. Also adds a <i>rowNum</i> column.
<i>airlineWithIndex</i>	The same data as the <i>airlineWithIntCol</i> table, but with a single clustered index using the <i>rowNum</i> column.
<i>airlineWithPageComp</i>	The same data as the <i>airlineWithIndex</i> table, but with page compression enabled. Also adds two columns, <i>CRSDepHour</i> and <i>Late</i> , which are computed from <i>CRSDepTime</i> and <i>ArrDelay</i> .

TABLE NAME	DESCRIPTION
<i>airlineWithRowComp</i>	The same data as the <i>airlineWithIndex</i> table, but with row compression enabled. Also adds two columns, <i>CRSDepHour</i> and <i>Late</i> , which are computed from <i>CRSDepTime</i> and <i>ArrDelay</i> .
<i>airlineColumnar</i>	A columnar store with a single clustered index. This table is populated with data from a cleaned up csv file.

Each test consisted of these steps:

1. R garbage collection was induced before each test.
2. A logistic regression model was created based on the table data. The value of *rowsPerRead* for each test was set to 500000.
3. Scores were generated using the trained model
4. Each test was run six times. The time of the first run (the "cold run") was dropped. To allow for the occasional outlier, the **maximum** time among the remaining five runs was also dropped. The average of the four remaining runs was taken to compute the average elapsed runtime of each test.
5. The tests were run using the *reportProgress* parameter with the value 3 (= report timings and progress). Each output file contains information regarding the time spent in IO, transition time, and compute time. These times are useful for troubleshooting and diagnosis.
6. The console output was also directed to a file in the output directory.
7. The test scripts processed the times in these files to compute the average time over runs.

For example, the following results are the times from a single test. The main timings of interest are **Total read time** (IO time) and **Transition time** (overhead in setting up processes for computation).

### Sample timings

```

Running IntCol Test. Using airlineWithIntCol table.
run 1 took 3.66 seconds
run 2 took 3.44 seconds
run 3 took 3.44 seconds
run 4 took 3.44 seconds
run 5 took 3.45 seconds
run 6 took 3.75 seconds

Average Time: 3.4425
metric time pct
1 Total time 3.4425 100.00
2 Overall compute time 2.8512 82.82
3 Total read time 2.5378 73.72
4 Transition time 0.5913 17.18
5 Total non IO time 0.3134 9.10

```

We recommend that you download and modify the test scripts to help you troubleshoot issues with R Services or with RevoScaleR functions.

### Test results (all)

This section compares before and after results for each of the tests.

#### Data size with compression and a columnar table store

The first test compared the use of compression and a columnar table to reduce the size of the data.

TABLE NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED	% SAVING (RESERVED)
<i>airlineWithIndex</i>	10000000	2978816 KB	2972160 KB	6128 KB	528 KB	0
<i>airlineWithPageComp</i>	10000000	625784 KB	623744 KB	1352 KB	688 KB	79%
<i>airlineWithRowComp</i>	10000000	1262520 KB	1258880 KB	2552 KB	1088 KB	58%
<i>airlineColumnar</i>	9999999	201992 KB	201624 KB	n/a	368 KB	93%

## Conclusions

The greatest reduction in data size was achieved by applying a columnstore index, followed by page compression.

### Effects of compression

This test compared the benefits of row compression, page compression, and no compression. A model was trained using `rxLinMod` on data from three different data tables. The same formula and query was used for all tables.

TABLE NAME	TEST NAME	NUMTASKS	AVERAGE TIME
<i>airlineWithIndex</i>	NoCompression	1	5.6775
	NoCompression - parallel	4	5.1775
<i>airlineWithPageComp</i>	PageCompression	1	6.7875
	PageCompression - parallel	4	5.3225
<i>airlineWithRowComp</i>	RowCompression	1	6.1325
	RowCompression - parallel	4	5.2375

## Conclusions

Compression alone does not seem to help. In this example, the increase in CPU to handle compression compensates for the decrease in IO time.

However, when the test is run in parallel by setting `numTasks` to 4, the average time decreases.

For larger data sets, the effect of compression may be more noticeable. Compression depends on the data set and values, so experimentation may be needed to determine the effect compression has on your data set.

### Effect of Windows power plan options

In this experiment, `rxLinMod` was used with the *airlineWithIntCol* table. The Windows Power Plan was set to either **Balanced** or **High performance**. For all tests, `numTasks` was set to 1. The test was run six times, and was performed twice under both power options to investigate the variability of results.

**High performance** power option:

TEST NAME	RUN #	ELAPSED TIME	AVERAGE TIME
IntCol	1	3.57 seconds	
	2	3.45 seconds	
	3	3.45 seconds	
	4	3.55 seconds	
	5	3.55 seconds	
	6	3.45 seconds	
			3.475
IntCol	1	3.45 seconds	
	2	3.53 seconds	
	3	3.63 seconds	
	4	3.49 seconds	
	5	3.54 seconds	
	6	3.47 seconds	
			3.5075

**Balanced** power option:

TEST NAME	RUN #	ELAPSED TIME	AVERAGE TIME
IntCol	1	3.89 seconds	
	2	4.15 seconds	
	3	3.77 seconds	
	4	5 seconds	
	5	3.92 seconds	
	6	3.8 seconds	
			3.91
IntCol	1	3.82 seconds	
	2	3.84 seconds	

TEST NAME	RUN #	ELAPSED TIME	AVERAGE TIME
	3	3.86 seconds	
	4	4.07 seconds	
	5	4.86 seconds	
	6	3.75 seconds	
			3.88

## Conclusions

The execution time is more consistent and faster when using the Windows **High performance** power plan.

### Using integer vs. strings in formulas

This test assessed the impact of modifying the R code to avoid a common problem with string factors. Specifically, a model was trained using `rxLinMod` using two tables: in the first, factors are stored as strings; in the second table, factors are stored as integers.

- For the *airline* table, the [DayOfWeek] column contains strings. The `colInfo` parameter was used to specify the factor levels (Monday, Tuesday, ...)
- For the *airlineWithIndex* table, [DayOfWeek] is an integer. The `colInfo` parameter was not specified.
- In both cases, the same formula was used: `ArrDelay ~ CRSDepTime + DayOfWeek`.

TABLE NAME	TEST NAME	AVERAGE TIME
<i>Airline</i>	<i>FactorCol</i>	10.72
<i>airlineWithIntCol</i>	<i>IntCol</i>	3.4475

## Conclusions

There is a clear benefit when using integers rather than strings for factors.

### Avoiding transformation functions

In this test, a model was trained using `rxLinMod`, but the code was changed between the two runs:

- In the first run, a transformation function was applied as part of model building.
- In the second run, the feature values were precomputed and available, so that no transformation function was required.

TEST NAME	AVERAGE TIME
WithTransformation	5.1675
WithoutTransformation	4.7

## Conclusions

Training time was shorter when **not** using a transformation function. In other words, the model was trained faster when using columns that are pre-computed and persisted in the table.

The savings is expected to be greater if there were many more transformations and the data set were larger (> 100M).

### Using columnar store

This test assessed the performance benefits of using a columnar data store and index. The same model was trained using `rxLinMod` and no data transformations.

- In the first run, the data table used a standard row store.
- In the second run, a column store was used.

TABLE NAME	TEST NAME	AVERAGE TIME
<i>airlineWithIndex</i>	RowStore	4.67
<i>airlineColumnar</i>	ColStore	4.555

### Conclusions

Performance is better with the columnar store than with the standard row store. A significant difference in performance can be expected on larger data sets (> 100 M).

### Effect of using the cube parameter

The purpose of this test was to determine whether the option in RevoScaleR for using the precomputed `cube` parameter can improve performance. A model was trained using `rxLinMod`, using this formula:

```
ArrDelay ~ Origin:DayOfWeek + Month + DayofMonth + CRSDepTime
```

In the table, the factors *DayOfWeek* is stored as a string.

TEST NAME	CUBE PARAMETER	NUMTASKS	AVERAGE TIME	SINGLE-ROW PREDICT (ARRDELAY_PRED)
CubeArgEffect	<code>cube = F</code>	1	91.0725	9.959204
		4	44.09	9.959204
	<code>cube = T</code>	1	21.1125	9.959204
		4	8.08	9.959204

### Conclusions

The use of the cube parameter argument clearly improves performance.

### Effect of changing maxDepth for rxDTree models

In this experiment, the `rxDTree` algorithm was used to create a model on the *airlineColumnar* table. For this test *numTasks* was set to 4. Several different values for *maxDepth* were used to demonstrate how altering tree depth affects run time.

TEST NAME	MAXDEPTH	AVERAGE TIME
TreeDepthEffect	1	10.1975
	2	13.2575

TEST NAME	MAXDEPTH	AVERAGE TIME
	4	19.27
	8	45.5775
	16	339.54

## Conclusions

As the depth of the tree increases, the total number of nodes increases exponentially. The elapsed time for creating the model also increased significantly.

### Prediction on a stored model

The purpose of this test was to determine the performance impacts on scoring when the trained model is saved to a SQL Server table rather than generated as part of the currently executing code. For scoring, the stored model is loaded from the database and predictions are created using a one-row data frame in memory (local compute context).

The test results show the time to save the model, and the time taken to load the model and predict.

TABLE NAME	TEST NAME	AVERAGE TIME (TO TRAIN MODEL)	TIME TO SAVE/LOAD MODEL
airline	SaveModel	21.59	2.08
airline	LoadModelAndPredict		2.09 (includes time to predict)

## Conclusions

Loading a trained model from a table is clearly a faster way to do prediction. We recommend that you avoid creating the model and performing scoring all in the same script.

## Case study: Optimization for the resume-matching task

The resume-matching model was developed by Microsoft data scientist Ke Huang to test the performance of R code in SQL Server, and by doing so help data scientists create scalable, enterprise-level solutions.

### Methods

Both the RevoScaleR and MicrosoftML packages were used to train a predictive model in a complex R solution involving large datasets. SQL queries and R code were identical in all tests. Tests were conducted on a single Azure VM with SQL Server installed. The author then compared scoring times with and without the following optimizations provided by SQL Server:

- In-memory tables
- Soft-NUMA
- Resource Governor

To assess the effect of soft-NUMA on R script execution, the data science team tested the solution on an Azure virtual machine with 20 physical cores. On these physical cores, four soft-NUMA nodes were created automatically, such that each node contained five cores.

CPU affinization was enforced in the resume-matching scenario, to assess the impact on R jobs. Four **SQL resource pools** and four **external resource pools** were created, and CPU affinity was specified to ensure that the same set of CPUs would be used in each node.

Each of the resource pools was assigned to a different workload group, to optimize hardware utilization. The reason is that Soft-NUMA and CPU affinity cannot divide physical memory in the physical NUMA nodes; therefore, by definition all soft NUMA nodes that are based on the same physical NUMA node must use memory in the same OS memory block. In other words, there is no memory-to-processor affinity.

The following process was used to create this configuration:

1. Reduce the amount of memory allocated by default to SQL Server.
2. Create four new pools for running the R jobs in parallel.
3. Create four workload groups such that each workload group is associated with a resource pool.
4. Restart Resource Governor with the new workload groups and assignments.
5. Create a user-defined classifier function (UDF) to assign different tasks on different workload groups.
6. Update the Resource Governor configuration to use the function for appropriate workload groups.

## Results

The configuration that had the best performance in the resume-matching study was as follows:

- Four internal resource pools (for SQL Server)
- Four external resource pools (for external script jobs)
- Each resource pool is associated with a specific workload group
- Each resource pool is assigned to different CPUs
- Maximum internal memory usage (for SQL Server) = 30%
- Maximum memory for use by R sessions = 70%

For the resume-matching model, external script use was heavy and there were no other database engine services running. Therefore, the resources allocated to external scripts were increased to 70%, which proved the best configuration for script performance.

This configuration was arrived at by experimenting with different values. If you use different hardware or a different solution, the optimum configuration might be different. Always experiment to find the best configuration for your case!

In the optimized solution, 1.1 million rows of data (with 100 features) were scored in under 8.5 seconds on a 20-core computer. Optimizations significantly improved the performance in terms of scoring time.

The results also suggested that the **number of features** had a significant impact on the scoring time. The improvement was even more prominent when more features were used in the prediction model.

We recommend that you read this blog article and the accompanying tutorial for a detailed discussion.

- [Optimization tips and tricks for machine learning in SQL Server](#)

Many users have noted that there is a small pause as the R (or Python) runtime is loaded for the first time. For this reason, as described in these tests, the time for the first run is often measured but later discarded.

Subsequent caching might result in notable performance differences between the first and second runs. There is also some overhead when data is moved between SQL Server and the external runtime, particularly if data is passed over the network rather than being loaded directly from SQL Server.

For all these reasons, there is no single solution for mitigating this initial loading time, as the performance impact varies significantly depending on the task. For example, caching is performed for single-row scoring in batches; hence, successive scoring operations are much faster and neither the model nor the R runtime is reloaded. You

can also use [native scoring](#) to avoid loading the R runtime entirely.

For training large models, or scoring in large batches, the overhead might be minimal in comparison to the gains from avoiding data movement or from streaming and parallel processing. See these recent blogs and samples for additional performance guidance:

- [Loan classification using SQL Server 2016 R Services](#)
- [Early customer experiences with R Services](#)
- [Using R to detect fraud at 1 million transactions per second](#)

## Resources

The following are links to information, tools, and scripts used in the development of these tests.

- Performance testing scripts and links to the data: [Sample data and scripts for SQL Server optimizations study](#)
- Article describing the resume-matching solution: [Optimization tip and tricks for SQL Server R Services](#)
- Scripts used in SQL optimization for resume-matching solution: [GitHub repository](#)

### Learn about Windows server management

- [How to determine the appropriate page file size for 64-bit versions of Windows](#)
- [Understanding NUMA](#)
- [How SQL Server supports NUMA](#)
- [Soft NUMA](#)

### Learn about SQL Server optimizations

- [Reorganize and Rebuild Indexes](#)
- [Introduction to memory-optimized tables](#)
- [Demonstration: Performance improvement of in-memory OLTP](#)
- [Data compression](#)
- [Enable Compression on a Table or Index](#)
- [Disable Compression on a Table or Index](#)

### Learn about managing SQL Server

- [Monitor and Tune for Performance](#)
- [Resource Governor](#)
- [Introducing Resource Governor](#)
- [Resource Governance for R Services](#)
- [How to Create a resource pool for R](#)
- [An example of configuring Resource Governor](#)

### Tools

- [DISKSPD storage load generator/performance test tool](#)
- [FSUtil utility reference](#)

## Other articles in this series

[Performance tuning for R – introduction](#)

[Performance tuning for R - SQL Server configuration](#)

[Performance tuning for R - R code and data optimization](#)

[Performance Tuning - case study results](#)

# Using R code profiling functions

4/12/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

In addition to using SQL Server resources and tools to monitor R script execution, you can use performance tools provided by other R packages to get more information about internal function calls. This topic provides a list of some basic resources to get you started. For expert guidance, we recommend the chapter on [Performance](#) in the book ""Advanced R""", by Hadley Wickham.

## Using RPROF

*rprof* is a function included in the base package **utils**, which is loaded by default. One advantage of *rprof* is that it performs sampling, thus lessening the performance load from monitoring.

To use R profiling in your code, you call this function and specify its parameters, including the name of the location of the log file that will be written. See the help for *rprof* for details.

In general, the *rprof* function works by writing out the call stack to a file, at specified intervals. You can then use the *summaryRprof* function to process the output file.

Profiling can be turned on and off in your code. To turn profiling on, suspend profiling, and then restart it, you would use a sequence of calls to *rprof*:

1. Specify profiling output file.

```
varOutputFile <- "C:/TEMP/run001.log")
Rprof(varOutputFile)
```

2. Turn off profiling

```
Rprof(NULL)
```

3. Restart profiling

```
Rprof(append=TRUE)
```

### NOTE

Using this function requires that Windows Perl be installed on the computer where code is run. Therefore, we recommend that you profile code during development in an R environment, and then deploy the debugged code to SQL Server.

## R System Functions

The R language includes many base package functions for returning the contents of system variables.

For example, as part of your R code, you might use `Sys.timezone` to get the current time zone, or `Sys.Time` to get the system time from R.

To get information about individual R system functions, type the function name as the argument to the R `help()` function from an R command prompt.

```
help("Sys.time")
```

## Debugging and Profiling in R

The documentation for Microsoft R Open, which is installed by default, includes a manual on developing extensions for the R language that discusses profiling and debugging in detail.

The chapter is also available online: <https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Debugging>

### Location of R help files

C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R\_SERVICES\doc\manual

# Configure and manage machine learning components in SQL Server

5/30/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides links to more detailed information about how to configure a server to support machine learning services with SQL Server in these products:

- [SQL Server 2016 R Services \(In-Database\)](#)
- [SQL Server 2017 Machine Learning Services \(In-Database\)](#)

## NOTE

This content was originally written for the SQL Server 2016 release, which supported only the R language.

In SQL Server 2017, support for Python has been added, but the underlying architecture and services framework is the same. Therefore, you can configure security, memory, resource governance and other options to support execution of Python scripts, the same way that you would for R scripts.

However, because support for Python is a new feature, detailed information about potential optimizations for the Python workload is not available yet. Please check back later.

## R Package Management

These articles describe how to install new R packages on the SQL Server instance, manage R package libraries, and restore package libraries after a database restore.

- [Default R and Python packages in SQL Server](#)
- [Installing New R Packages](#)
- [Enable Package Management for an Instance using Database Roles](#)
- [Create a Local Package Repository using miniCRAN](#)
- [Determine Which R Packages are Installed on SQL Server](#)
- [Synchronizing R Packages between SQL Server and the File System](#)
- [R Packages Installed in User Libraries](#)

## Service Configuration

These articles describe how to make changes to the underlying service architecture and how to manage security principals associated with the extensibility service.

- [Security Considerations](#)
- [Modify the User Account Pool for SQL Server R Services](#)
- [Configure and Manage Advanced Analytics Extensions](#)
- [Enable Package Management for an Instance using Database Roles](#)
- [Performance Tuning for R Services](#)

## Resource Governance

These articles describe how to implement resource management for R or Python jobs using the Resource Governor feature available in Enterprise Edition.

- [Resource Governance for R Services](#)
- [How to Create a Resource Pool for R](#)

Also see:

- [Monitor R Using Custom SSMS Reports](#)

## Initial Setup

Additional help related to initial setup and configuration can be found in these articles:

- [Upgrade and Installation FAQ](#)
- [Security Considerations](#)
- [Known Issues for R Services](#)

# Advanced configuration options for SQL Server Machine Learning Services

4/12/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes changes that you can make after setup, to modify the configuration of the external script runtime and other services associated with machine learning in SQL Server.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Provision additional user accounts for machine learning

External script processes in SQL Server run in the context of low-privilege local user accounts. Running these processes in individual low-privilege accounts has the following benefits:

- Reduces privileges of the external script runtime processes on the SQL Server computer
- Provides isolation between sessions of an external runtime such as R or Python.

As part of setup, a new Windows *user account pool* is created that contains the local user accounts required for running external runtime processes, such as R or Python. You can modify the number of users as needed to support machine learning tasks.

Additionally, your database administrator must give this group permission to connect to any instance where machine learning has been enabled. For more information, see [Modify the user account pool for SQL Server Machine Learning Services](#).

To protect sensitive resources on the SQL Server, you can define an access control list (ACL) for this group. By specifying resources that the group is denied access to, you can prevent access by external processes such as the R or Python runtimes.

- The user account pool is linked to a specific instance. A separate pool of worker accounts is needed for each instance on which machine learning has been enabled. Accounts cannot be shared between instances.
- User account names in the pool are of the format `SQLInstanceNamenn`. For example, if you are using the default instance for machine learning, the user account pool supports account names such as `MSSQLSERVER01`, `MSSQLSERVER02`, and so forth.
- The size of the user account pool is static and the default value is 20. The number of external runtime sessions that can be launched simultaneously is limited by the size of this user account pool. To change this limit, an administrator should use SQL Server Configuration Manager.

For more information about how to make changes to the user account pool, see [Modify the user account pool for SQL Server Machine Learning Services](#).

## Manage memory used by external script processes

By default, the external script runtimes for machine learning are limited to no more than 20% of total machine memory. It depends on your system, but in general, you might find this limit inadequate for serious machine learning tasks such as training a model or predicting on many rows of data.

To support machine learning, an administrator can increase this limit. When you do so, you might need to reduce

the amount of memory reserved for SQL Server or for other services. You should also consider using Resource Governor to define an external resource pool or pools, so that you can allocate specific resource pools to R or Python jobs.

For more information, see [Resource governance for machine learning](#).

## Modify the Launchpad service account

A separate SQL Server Trusted Launchpad service is created for each instance on which you have configured the machine learning services.

By default, the Launchpad is configured to run using the account, NT Service\MSQLLaunchpad, which is provisioned with all necessary permissions to run R scripts. However, if you change this account, the Launchpad might not be able to start or to access the SQL Server instance where external scripts should be run.

If you modify the service account, be sure to use the **Local Security Policy** application and update the permissions on each service account to include these permissions:

- Adjust memory quotas for a process (SeIncreaseQuotaPrivilege)
- Bypass traverse checking (SeChangeNotifyPrivilege)
- Log on as a service (SeServiceLogonRight)
- Replace a process-level token (SeAssignPrimaryTokenPrivilege)

For more information about permissions required to run SQL Server services, see [Configure Windows Service Accounts and Permissions](#).

## Change advanced service options

In early versions of SQL Server 2016 R Services, you could change some properties of the service by editing the R Services (In-Database) configuration file.

However, this file is no longer used for changing configurations. We recommend that you use SQL Server Configuration Manager to effect changes to service configuration, such as the service account and number of users.

### To modify Launchpad configuration

1. Open [SQL Server Configuration Manager](#).
2. Right-click SQL Server Launchpad and select **Properties**.
  - To change the service account, click the **Log On** tab.
  - To increase the number of users, click the **Advanced** tab.

### To modify debug settings

A few properties can only be changed by using the Launchpad's configuration file, which might be useful in limited cases, such as debugging. The configuration file is created during SQL Server setup and by default is saved as a plain text file in the following location: <instance path>\binn\rlauncher.config

You must be an administrator on the computer that is running SQL Server to make changes to this file. If you edit the file, we recommend that you make a backup copy before saving changes.

The following table lists the advanced settings for SQL Server 2017, with the permissible values.

SETTING NAME	TYPE	DESCRIPTION
--------------	------	-------------

SETTING NAME	TYPE	DESCRIPTION
JOB_CLEANUP_ON_EXIT	Integer	<p>This is an internal setting only – do not change this value.</p> <p>Specifies whether the temporary working folder created for each external runtime session should be cleaned up after the session is completed. This setting is useful for debugging.</p> <p>Supported values are <b>0</b> (Disabled) or <b>1</b> (Enabled).</p> <p>The default is 1, meaning log files are removed on exit.</p>
TRACE_LEVEL	Integer	<p>Configures the trace verbosity level of MSSQLLAUNCHPAD for debugging purposes. This affects trace files in the path specified by the LOG_DIRECTORY setting.</p> <p>Supported values are: <b>1</b> (Error), <b>2</b> (Performance), <b>3</b> (Warning), <b>4</b> (Information).</p> <p>The default is 1, meaning output warnings only.</p>

All settings take the form of a key-value pair, with each setting on a separate line. For example, to change the trace level, you would add the line `Default: TRACE_LEVEL=4`.

## See Also

[Security considerations](#)

# Security considerations for machine learning in SQL Server

4/12/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article lists security considerations that the administrator or architect should bear in mind when using machine learning services.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Use a firewall to restrict network access

In a default installation, a Windows firewall rule is used to block all outbound network access from external runtime processes. Firewall rules should be created to prevent the external runtime processes from downloading packages or from making other network calls that could potentially be malicious.

If you are using a different firewall program, you can also create rules to block outbound network connection for external runtimes, by setting rules for the local user accounts or for the group represented by the user account pool.

We strongly recommend that you turn on Windows Firewall (or another firewall of your choice) to prevent unrestricted network access by the R or Python runtimes.

## Authentication methods supported for remote compute contexts

R Services (In-Database) supports both Windows Integrated Authentication and SQL logins when creating connections between SQL Server and a remote data science client.

For example, say you are developing an R solution on your laptop and want to perform computations on the SQL Server computer. You would create a SQL Server data source in R, by using the `rx` functions and defining a connection string based on your Windows credentials.

When you change the *compute context* from your laptop to the SQL Server computer, all R code is executed on the SQL Server computer, if your Windows account has the necessary permissions. Moreover, any SQL queries executed as part of the R code are run under your credentials as well.

The use of SQL logins is also supported in this scenario. However, this requires that the SQL Server instance be configured to allow mixed mode authentication.

### Implied authentication

In general, the SQL Server Trusted Launchpad starts the external script runtime and executes scripts under its own account. However, if the external runtime makes an ODBC call, the SQL Server Trusted Launchpad impersonates the credentials of the user that sent the command to ensure that the ODBC call does not fail. This is called *implied authentication*.

## IMPORTANT

For implied authentication to succeed, the Windows users group that contains the worker accounts (by default, **SQLRUserGroup**) must have an account in the master database for the instance, and this account must be given permissions to connect to the instance.

The group **SQLRUserGroup** is also used when running Python scripts.

In general, we recommend that you move larger datasets into SQL Server beforehand, rather than try to read data using RODBC or another library. Also, use a SQL Server query or view as your primary data source, for better performance.

For example, you might get your training data (typically the largest data) from SQL Server and get a list of factors from an external source. You can define a linked server to get data from most ODBC data sources. For more information, see [Create a linked server](#).

To minimize dependency on ODBC calls to external data sources, you might also perform data engineering as a separate process, and save the results in a table, or use T-SQL. See this tutorial for a good example of data engineering in SQL vs. R: [Create data features using T-SQL](#).

## No support for encryption at rest

[Transparent Data Encryption \(TDE\)](#) is not supported for data sent to or received from the external script runtime. The reason is that R (or Python) runs outside the SQL Server process; thus, data used by the external runtime is not protected by the encryption features of the database engine. This behavior is no different than any other client running on the SQL Server computer that reads data from the database and makes a copy.

As a consequence, TDE **is not** applied to any data that you use in R or Python scripts, or to any data saved to disk, or to any persisted intermediate results. However, other types of encryption, such as Windows BitLocker encryption or third-party encryption applied at the file or folder level, still apply.

In the case of [Always Encrypted](#), external runtimes do not have access to the encryption keys; therefore, data cannot be sent to the scripts.

## Resources

For more information about managing the service, and about how to provision the user accounts used to execute machine learning scripts, see [Configure and manage Advanced Analytics Extensions](#).

For an explanation of the general security architecture, see:

- [Security overview for R](#)
- [Security overview for Python](#)

# Modify the user account pool for SQL Server machine learning

4/12/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

As part of the installation process for Machine Learning Services (In-Database), a new Windows *user account pool* is created to support execution of tasks by the SQL Server Trusted Launchpad service. The purpose of these worker accounts is to isolate concurrent execution of external scripts by different SQL users.

This article describes the default configuration, security and capacity for the worker accounts, and how to change the default configuration.

**Applies to:** SQL Server 2016 (13.x) R Services (In-Database), SQL Server 2017 Machine Learning Services (In-Database)

## Worker accounts used for external script execution

The Windows account group is created by SQL Server setup for each instance on which machine learning is installed and enabled.

- In a default instance, the group name is **SQLRUserGroup**. The name is the same whether you use R or Python or both.
- In a named instance, the default group name is suffixed with the instance name: for example, **SQLRUserGroupMyInstanceName**.

By default, the user account pool contains 20 user accounts. In most cases, 20 is more than adequate to support machine learning tasks, but you can change the number of accounts. The maximum number of accounts is 100.

- In a default instance, the individual accounts are named **MSSQLSERVER01** through **MSSQLSERVER20**.
- For a named instance, the individual accounts are named after the instance name: for example, **MyInstanceName01** through **MyInstanceName20**.

If more than one instance uses machine learning, the computer will have multiple user groups. A group cannot be shared across instances.

### How to change the number of worker accounts

To modify the number of users in the account pool, you must edit the properties of the SQL Server Trusted Launchpad service as described below.

Passwords associated with each user account are generated at random, but you can change them later, after the accounts have been created.

1. Open SQL Server Configuration Manager and select **SQL Server Services**.
2. Double-click the SQL Server Launchpad service and stop the service if it is running.
3. On the **Service** tab, make sure that the Start Mode is set to Automatic. External scripts cannot start when the Launchpad is not running.
4. Click the **Advanced** tab and edit the value of **External Users Count** if necessary. This setting controls how many different SQL users can run external script sessions concurrently. The default is 20 accounts. The maximum number of users is 100.
5. Optionally, you can set the option **Reset External Users Password** to **Yes** if your organization has a policy

that requires changing passwords on a regular basis. Doing this will regenerate the encrypted passwords that Launchpad maintains for the user accounts. For more information, see [Enforcing Password Policy](#).

6. Restart the Launchpad service.

## Managing machine learning workloads

The number of accounts in this pool determines how many external script sessions can be active simultaneously. By default, 20 accounts are created, meaning that 20 different users can have active R or Python sessions at one time. You can increase the number of worker accounts, if you expect to run more than 20 concurrent scripts.

When the same user executes multiple external scripts concurrently, all the sessions run by that user use the same worker account. For example, a single user might have 100 different R scripts running concurrently, as long as resources permit, but all scripts would run using a single worker account.

The number of worker accounts that you can support, and the number of concurrent sessions that any single user can run, is limited only by server resources. Typically, memory is the first bottleneck that you will encounter when using the R runtime.

The resources that can be used by Python or R scripts are governed by SQL Server. We recommend that you monitor resource usage using SQL Server DMVs, or look at performance counters on the associated Windows job object, and adjust server memory use accordingly. If you have SQL Server Enterprise Edition, you can allocate resources used for running external scripts by configuring an [external resource pool](#).

For additional information about managing machine learning task capacity, see these articles:

- [SQL Server configuration for R Services](#)
- [Performance case study for R Services](#)

## Security

Each user group is associated with the SQL Server Trusted Launchpad service on a specific instance and cannot support R jobs that run on other instances.

For each worker account, while the session is active, a temporary folder is created to store the script objects, intermediate results, and other information used by R and SQL Server during R script execution. These working files, located under the ExtensibilityData folder, are access-restricted to administrators, and are cleaned up by SQL Server after the script completes.

For more information, see [Security Overview](#).

### Enforcing password policy

If your organization has a policy that requires changing passwords on a regular basis, you may need to force the Launchpad service to regenerate the encrypted passwords that Launchpad maintains for its worker accounts.

To enable this setting and force password refresh, open the **Properties** pane for the Launchpad service in SQL Server Configuration Manager, click **Advanced**, and change **Reset External Users Password** to **Yes**. When you apply this change, the passwords will immediately be regenerated for all user accounts. To use R script after this change, you must restart the Launchpad service, at which time it will read the newly generated passwords.

To reset passwords at regular intervals, you can either set this flag manually or use a script.

### Additional permission required to support remote compute contexts

By default, the group of R worker accounts does **not** have login permissions on the SQL Server instance with which it is associated. This can be a problem if any R users connect to SQL Server from a remote client to run R scripts, or if a script uses ODBC to get additional data.

To ensure that these scenarios are supported, the database administrator must provide the group of R worker

accounts with permission to log into the SQL Server instance where R scripts will be run (**Connect to** permissions). This is referred to as *implied authentication*, and enables SQL Server to run the R scripts using the credentials of the remote user.

**NOTE**

This limitation does not apply if you use SQL logins to run R scripts from a remote workstation, because the SQL login credentials are explicitly passed from the R client to the SQL Server instance and then to ODBC.

### How to enable implied authentication

1. Open SQL Server Management Studio as an administrator on the instance where you will run R or Python code.
2. Run the following script. Be sure to edit the user group name, if you changed the default, and the computer and instance name.

```
USE [master]
GO

CREATE LOGIN [computername\SQLRUserGroup] FROM WINDOWS WITH DEFAULT_DATABASE=[master],
DEFAULT_LANGUAGE=[language]
GO
```

# Add SQLRUserGroup as a database user

4/12/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article explains how to give the group of worker accounts used by machine learning services in SQL Server the permissions required to connect to the database and run R or Python jobs on behalf of the user.

## What is SQLRUserGroup?

During setup of Machine Learning Services (In-Database) or R Services (In-Database), new Windows user accounts are created to support execution of R or Python script tasks under the security token of the SQL Server Trusted Launchpad service.

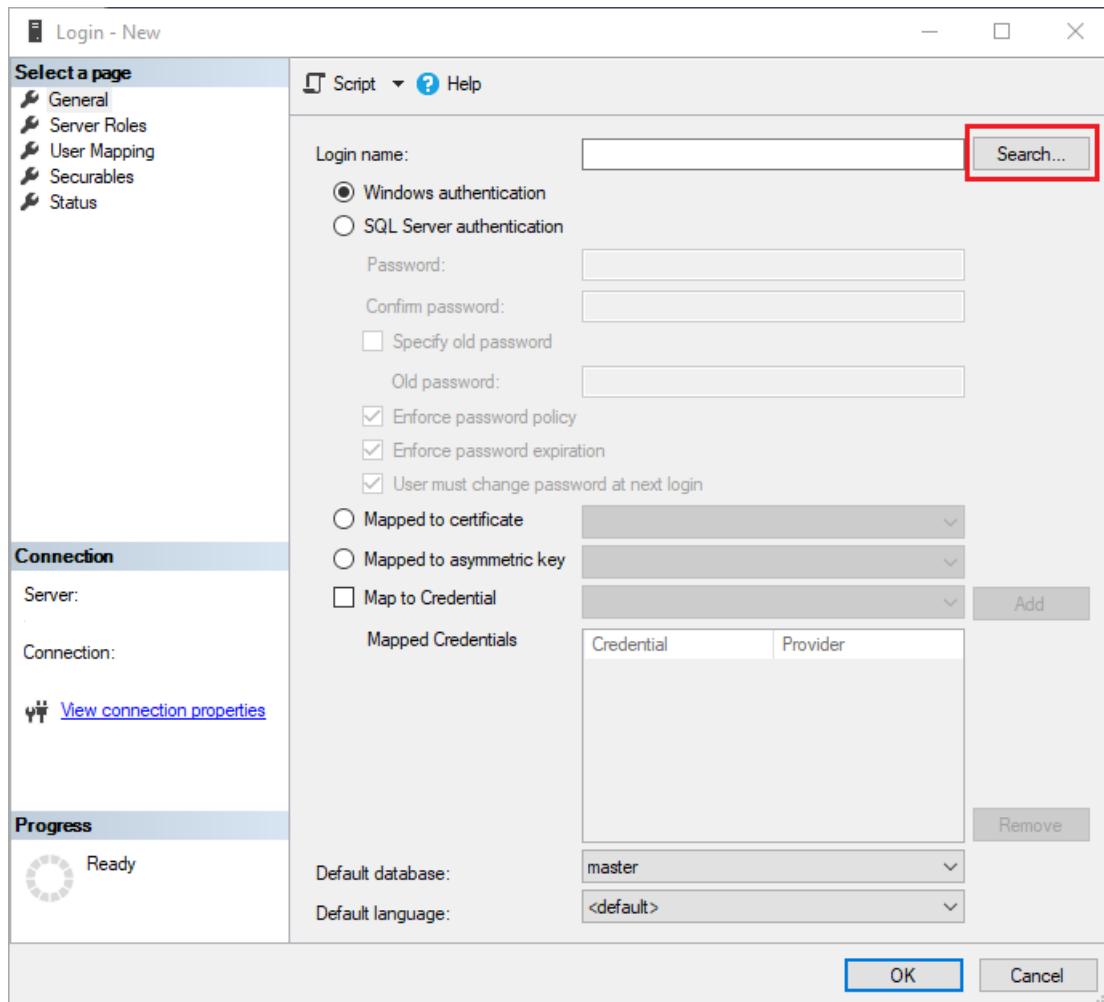
You can view these accounts in the Windows user group **SQLRUserGroup**. By default, 20 worker accounts are created, which is usually more than enough for running machine learning jobs.

When a user sends a machine learning script from an external client, SQL Server activates an available worker account, maps it to the identity of the calling user, and runs the script on behalf of the user. This new service of the database engine supports the secure execution of external scripts, called *implied authentication*.

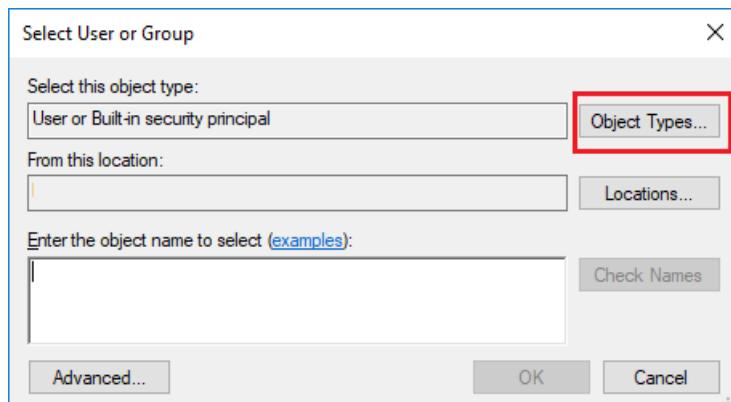
However, if you need to run R or Python scripts from a remote data science client, and you are using Windows authentication, you must give these worker accounts permission to sign in to the SQL Server instance on your behalf.

## Add SQLRUserGroup as a SQL Server login

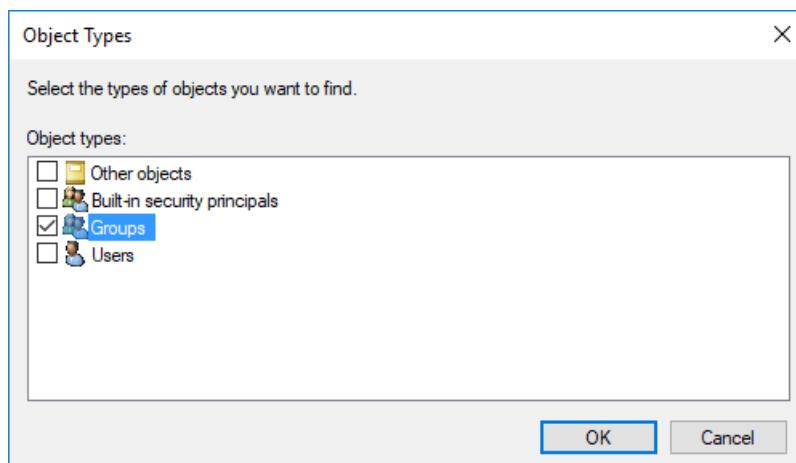
1. In SQL Server Management Studio, in Object Explorer, expand **Security**, right-click **Logins**, and select **New Login**.
2. In the **Login - New** dialog box, select **Search**. (Don't type anything in the box yet.)



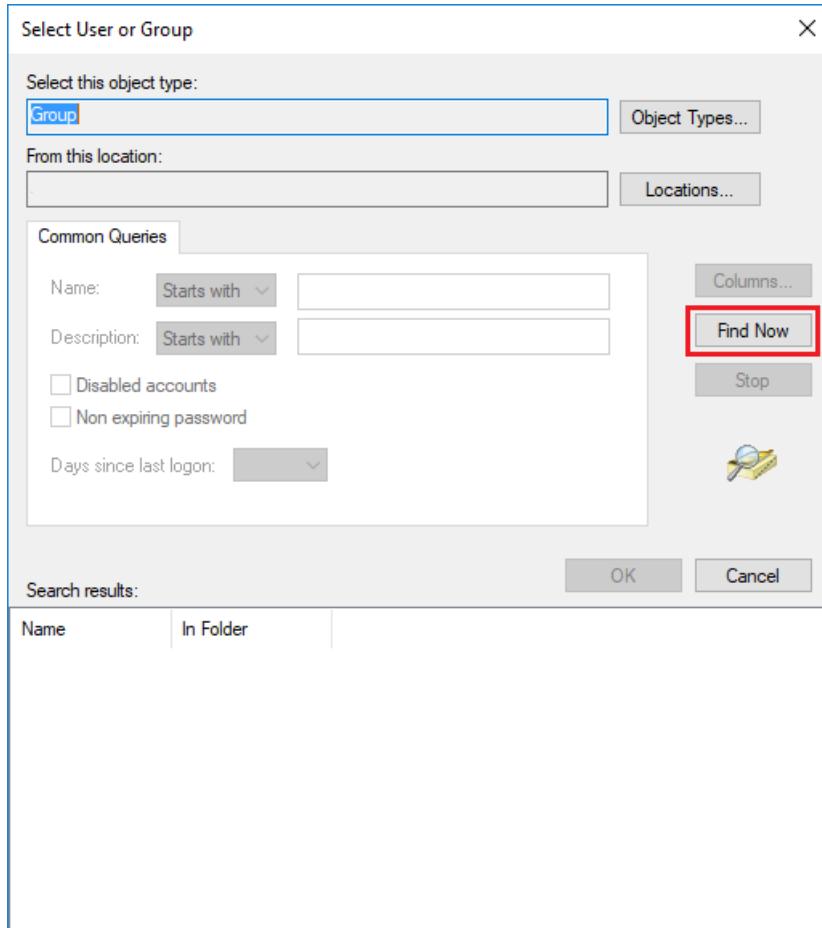
3. In the **Select User or Group** box, click the **Object Types** button.



4. In the **Object Types** dialog box, select **Groups**. Clear all other check boxes.

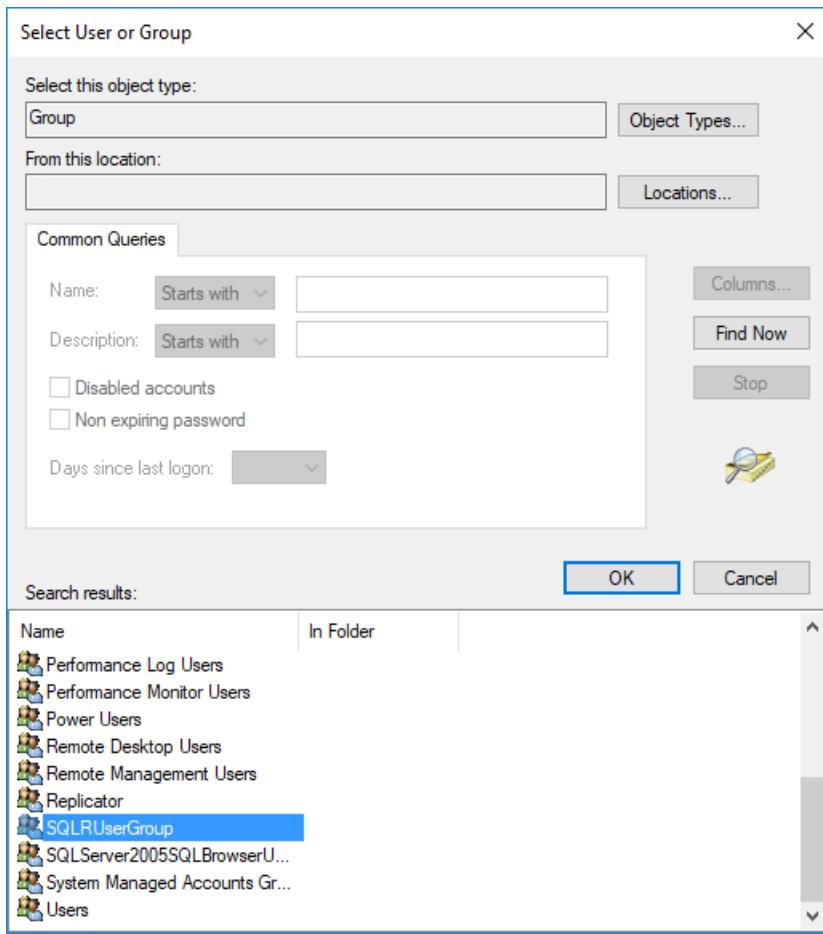


5. Click **Advanced**, verify that the location to search is the current computer, and then click **Find Now**.



6. Scroll through the list of group accounts on the server until you find one beginning with `SQLRUserGroup`.

- The name of the group that's associated with the Launchpad service for the *default instance* is always **SQLRUserGroup**, regardless of whether you installed R or Python or both. Select this account for the default instance only.
- If you are using a *named instance*, the instance name is appended to the name of the default worker group name, `SQLRUserGroup`. Hence, if your instance is named "MLTEST", the default user group name for this instance would be **SQLRUserGroupMLTest**.



7. Click **OK** to close the advanced search dialog box.

**IMPORTANT**

Be sure you've selected the correct account for the instance. Each instance can use only its own Launchpad service and the group created for that service. Instances cannot share a Launchpad service or worker accounts.

8. Click **OK** once more to close the **Select User or Group** dialog box.
9. In the **Login - New** dialog box, click **OK**. By default, the login is assigned to the **public** role and has permission to connect to the database engine.

## Change the number of worker accounts in SQLRUserGroup

If you intend to make heavy use of machine learning, you can increase the number of accounts used to run external scripts, as described in this article:

- [Modify the user account pool for machine learning](#)

By default, 20 accounts are created, which supports 20 concurrent sessions. Parallelized tasks do not consume additional accounts. For example, if a user runs a scoring task that uses parallel processing, the same worker account is reused for all threads.

# Deploy and consume analytics using mrsdeploy

4/11/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Microsoft R Server includes an operationalization feature, **mrsdeploy**, that supports these tasks:

- Publishing and managing R and Python models and code in the form of web services
- Consuming these services within client applications

This topic provides information about how to enable and configure the feature.

For general information about scenarios supported by **mrsdeploy**, see [Operationalization with R Server](#).

## Using mrsdeploy for operationalization

The word *operationalization* can mean many things:

- The ability to publish models to a web service for use by applications
- Support for scalable or distributed computing
- Develop once, deploy many times
- Fast scoring, for both single-row and batch scoring

If you have installed Machine Learning Services with SQL Server, *operationalization* is a matter of wrapping your R or Python code in a stored procedure. Any application can then call the stored procedure to retrain a model, generate scores, or create reports. You can also automate jobs using existing scheduling mechanisms in SQL Server.

However, Microsoft R Server provides a different mechanism for deployment support, using web services that support publishing of R jobs, and an administrative utility for running distributed R jobs. Microsoft R Server uses the functions in the **mrsdeploy** package to establish a session with remote compute nodes and execute R code in a console application.

This deployment feature of R Server provides these benefits:

- Role-based access control to analytical web services

Determine who can publish, update, and delete their own web services, those who can also update and delete the web services published by other users, and who can only list and consume web services.

- Faster scoring

You can use realtime scoring with a supported R model object to improve the speed of scoring operations.

- Asynchronous batch consumption

Web services that call for large input data can now be consumed asynchronously via batch execution.

- Autoscaling of a grid of web and compute nodes

A script template is provided to let you easily spin up a set of R Server VMs in Azure, and then configure them as a grid for operationalizing analytics and remote execution. This grid can be scaled up or down based on CPU usage.

- Asynchronous remote execution

Now supported using the **mrsdeploy** R package. To continue working in your development environment during remote script execution, execute your R script asynchronously using the `async` parameter. This is particularly useful when you are running scripts that have long execution times.

## Requirements and configuration

SQL Server 2017 CTP 2.0 and later includes this feature, which was previously available only with R Server, and not installed with SQL Server R Services. The **mrsdeploy** package is installed on the SQL Server computer, if you select the option to install **Microsoft Machine Learning Server**, from the **Shared Features** section of SQL Server setup.

Typically, we do not recommend that you install Machine Learning Server on the same computer that is running SQL Server Machine Learning Services. We recommend that you install **Microsoft Machine Learning Server** on a separate computer from SQL Server, and then configure the operationalization features on that computer.

However, if you need to install them together, follow these additional steps to successfully configure the service.

1. Install DotNetCore 1.1

If .NET Core was not installed as part of SQL Server, you must install it before beginning R Server setup.

2. Install Microsoft Machine Learning Server.

3. After completing setup of **Microsoft Machine Learning Server**, manually add the following registry key for **mrsdeploy**, which specifies the base folder for the R\_SERVER files.

- Create a new registry key `H_KEY_LOCAL_MACHINE\SOFTWARE\R_Server\Path`
- Set the value of the key to `"C:\Program Files\Microsoft SQL Server\140\R_SERVER"`.

4. When done, open the [Administrator Utility](#).

5. Continue to configure the **mrsdeploy** service as described here: [Configuration for administrators](#)

6. For more information, see [mrsdeploy functions](#).

# Managing and monitoring machine learning solutions

5/30/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes features in SQL Server Machine Learning Services that are relevant to database administrators who need to begin working with R and Python solutions.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Security

Database administrators must provide data access not just to data scientists but to a variety of report developers, business analysts, and business data consumers. The integration of R (and now Python) into SQL Server provides many benefits to the database administrator who supports the data science role.

- The architecture of R Services (In-Database) keeps your databases secure and isolates the execution of R sessions from the operation of the database instance.
- You can specify who has permission to execute the R scripts and ensure that the data used in R jobs is managed using the same security roles that are defined in SQL Server.
- The database administrator can use roles to manage the installation of R packages and the execution of R and Python scripts.

For more information, see these resources:

- [Security Considerations for the R Runtime in SQL Server](#)
- [R security overview](#)
- [Python security overview](#)
- [Default R and Python packages in SQL Server](#)

## Configuration and management

Database administrators must integrate competing projects and priorities into a single point of contact: the database server. They must support analytics while maintaining the health of operational and reporting data stores. The integration of machine learning with SQL Server provides many benefits to the database administrator, who increasingly serves a critical role in deploying an efficient infrastructure for data science.

- R and Python sessions are executed in a separate process to ensure that your server continues to run as usual even if the external script runtime has problems.
- Low privilege physical user accounts are used to contain and isolate external script activity.
- The DBA can use standard SQL Server resource management tools to control the amount of resources allocated to the R runtime, to prevent massive computations from jeopardizing the overall server performance.

For more information, see these resources:

- Resource governance for R Services
- Configure and manage Advanced Analytics Extensions

# Resource governance for machine learning in SQL Server

4/12/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides an overview of resource governance features in SQL Server that help allocate and balance resources used by R and Python scripts.

**Applies to:** SQL Server 2017 Machine Learning Services (In-Database) and SQL Server 2016 (13.x) R Services (In-Database)

## Goals of resource governance for machine learning

One known pain point with machine learning languages such as R and Python is that data is often moved outside the database to computers not controlled by IT. Another is that R is single-threaded, meaning that you can only work with the data available in memory.

SQL Server Machine Learning Services alleviates both these problems, and helps meet enterprise compliance requirements. It keeps advanced analytics inside the database, and supports increased performance over large datasets through features such as streaming and chunking operations. However, moving R and Python computations inside the databases can affect the performance of other services that use the database, including regular user queries, external applications, and scheduled database jobs.

This section provides information about how you can manage resources used by external runtimes, such as R and Python, to mitigate impact on other core database services. A database server environment typically is the hub for multiple dependent applications and services.

You can use [Resource Governor](#) to manage the resources used by the external runtimes for R and Python. For machine learning, resource governance involves these tasks:

- Identifying scripts that use excessive server resources.

The administrator needs to be able to terminate or throttle jobs that are consuming too many resources.

- Mitigating unpredictable workloads.

For example, if multiple machine learning jobs are running concurrently on the server, the resulting resource contention could lead to unpredictable performance or threaten completion of the workload. However, if resource pools are used, the jobs can be isolated from each other.

- Prioritizing workloads.

The administrator or architect needs to be able to specify workloads that must take precedence, or guarantee certain workloads to complete when there is resource contention.

## How to use Resource Governor to manage machine learning

You manage resources allocated to R or Python sessions by creating an *external resource pool*, and assigning workloads to the pool or pools. An external resource pool is a new type of resource pool introduced in SQL Server 2016 (13.x) to help manage the R runtime and other processes external to the database engine.

SQL Server supports three types of default resource pools:

- The *internal pool* represents the resources used by the SQL Server itself and cannot be altered or restricted.
- The *default pool* is a predefined user pool that you can use to modify resource use for the server as a whole. You can also define user groups that belong to this pool, to manage access to resources.
- The *default external pool* is a predefined user pool for external resources. Additionally, you can create new external resource pools and define user groups to belong to this pool.

In addition, you can create *user-defined resource pools* to allocate resources to the database engine or other applications, and create *user-defined external resource pools* to manage R and other external processes.

For a good introduction to terminology and general concepts, see [Resource Governor Resource Pool](#).

## Resource management walkthrough with Resource Governor

If you are new to Resource Governor, see this topic for a quick walkthrough of how to modify the instance default resources and create a new external resource pool: [Create a resource pool for external scripts](#)

You can use the *external resource pool* mechanism to manage the resources used by the following executables that are used in machine learning:

- Rterm.exe and satellite processes
- Python.exe and satellite processes
- BxlServer.exe and satellite processes
- Satellite processes launched by Launchpad

### NOTE

Direct management of the Launchpad service by using Resource Governor is not supported. That is because the SQL Server Trusted Launchpad is a trusted service that can by design host only launchers that are provided by Microsoft. Trusted launchers are configured to avoid consuming excessive resources.

We recommend that you manage satellite processes using Resource Governor and tune them to meet the needs of the individual database configuration and workload. For example, any individual satellite process can be created or destroyed on demand during execution.

## Disable external script execution

Support for external scripts is optional in SQL Server setup. Even after installing the machine learning features, the ability to execute external scripts is OFF by default, and you must manually reconfigure the property and restart the instance to enable script execution.

Therefore, if there is a resource issue that needs to be mitigated immediately, or a security issue, an administrator can immediately disable any external script execution by using [sp\\_configure \(Transact-SQL\)](#) and setting the property `external scripts enabled` to FALSE or 0.

## See also

[Managing and monitoring machine learning solutions](#)

[Create a resource pool for machine learning](#)

[Resource Governor resource pools](#)

# Create a resource pool for machine learning in SQL Server

4/12/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

This article describes how you can create a resource pool specifically for managing machine learning workloads in SQL Server. It assumes that you have already installed and enabled the machine learning features, and want to reconfigure the instance to support more fine-grained management of the resources used by external process such as R or Python.

The process includes multiple steps:

1. Review status of any existing resource pools. It is important that you understand what services are using existing resources.
2. Modify server resource pools.
3. Create a new resource pool for external processes.
4. Create a classification function to identify external script requests.
5. Verify that the new external resource pool is capturing R or Python jobs from the specified clients or accounts.

**Applies to:** SQL Server 2016 (13.x) R Services (In-Database) and SQL Server 2017 Machine Learning Services (In-Database)

## Review the status of existing resource pools

1. Use a statement such as the following to check the resources allocated to the default pool for the server.

```
SELECT * FROM sys.resource_governor_resource_pools WHERE name = 'default'
```

### Sample results

POOL_ID	NAME	MIN_CPU_PERCENT	MAX_CPU_PERCENT	MIN_MEMORY_PERCENT	MAX_MEMORY_PERCENT	CAP_CPU_PERCENT	MIN_IOPS_PER_VOLUME	MAX_IOPS_PER_VOLUME
2	default	0	100	0	100	100	0	0

2. Check the resources allocated to the default **external** resource pool.

```
SELECT * FROM sys.resource_governor_external_resource_pools WHERE name = 'default'
```

### Sample results

EXTERNAL_POOL_ID	NAME	MAX_CPU_PERCENT	MAX_MEMORY_PERCENT	MAX_PROCESSES	VERSION
2	default	100	20	0	2

3. Under these server default settings, the external runtime will probably have insufficient resources to

complete most tasks. To change this, you must modify the server resource usage as follows:

- Reduce the maximum computer memory that can be used by the database engine.
- Increase the maximum computer memory that can be used by the external process.

## Modify server resource usage

1. In Management Studio, run the following statement to limit SQL Server memory usage to **60%** of the value in the 'max server memory' setting.

```
ALTER RESOURCE POOL "default" WITH (max_memory_percent = 60);
```

2. Similarly, run the following statement to limit the use of memory by external processes to **40%** of total computer resources.

```
ALTER EXTERNAL RESOURCE POOL "default" WITH (max_memory_percent = 40);
```

3. To enforce these changes, you must reconfigure and restart Resource Governor as follows:

```
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

### NOTE

These are just suggested settings to start with; you should evaluate your machine learning tasks in light of other server processes to determine the correct balance for your environment and workload.

## Create a user-defined external resource pool

1. Any changes to the configuration of Resource Governor are enforced across the server as a whole and affect workloads that use the default pools for the server, as well as workloads that use the external pools.

Therefore, to provide more fine-grained control over which workloads should have precedence, you can create a new user-defined external resource pool. You should also define a classification function and assign it to the external resource pool. The **EXTERNAL** keyword is new.

Begin by creating a new *user-defined external resource pool*. In the following example, the pool is named **ds\_ep**.

```
CREATE EXTERNAL RESOURCE POOL ds_ep WITH (max_memory_percent = 40);
```

2. Create a workload group named **ds\_wg** to use in managing session requests. For SQL queries you'll use the default pool; for all external process queries will use the **ds\_ep** pool.

```
CREATE WORKLOAD GROUP ds_wg WITH (importance = medium) USING "default", EXTERNAL "ds_ep";
```

Requests are assigned to the default group whenever the request cannot be classified, or if there is any other classification failure.

For more information, see [Resource Governor Workload Group](#) and [CREATE WORKLOAD GROUP \(Transact-SQL\)](#).

## Create a classification function for machine learning

A classification function examines incoming tasks and determines whether the task is one that can be run using the current resource pool. Tasks that do not meet the criteria of the classification function are assigned back to the server's default resource pool.

1. Begin by specifying that a classifier function should be used by Resource Governor to determine resource pools. You can assign a **null** as a placeholder for the classifier function.

```
ALTER RESOURCE GOVERNOR WITH (classifier_function = NULL);
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

For more information, see [ALTER RESOURCE GOVERNOR \(Transact-SQL\)](#).

2. In the classifier function for each resource pool, define the type of statements or incoming requests that should be assigned to the resource pool.

For example, the following function returns the name of the schema assigned to the user-defined external resource pool if the application that sent the request is either 'Microsoft R Host' or 'RStudio'; otherwise it returns the default resource pool.

```
USE master
GO
CREATE FUNCTION is_ds_apps()
RETURNS sysname
WITH schemabinding
AS
BEGIN
 IF program_name() in ('Microsoft R Host', 'RStudio') RETURN 'ds_wg';
 RETURN 'default';
END;
GO
```

3. When the function has been created, reconfigure the resource group to assign the new classifier function to the external resource group that you defined earlier.

```
ALTER RESOURCE GOVERNOR WITH (classifier_function = dbo.is_ds_apps);
ALTER RESOURCE GOVERNOR WITH reconfigure;
GO
```

## Verify new resource pools and affinity

To verify that the changes have been made, you should check the configuration of server memory and CPU for each of the workload groups associated with these instance resource pools:

- the default pool for the SQL Server server
- the default resource pool for external processes
- the user-defined pool for external processes

1. Run the following statement to view all workload groups:

```
SELECT * FROM sys.resource_governor_workload_groups;
```

### Sample results

GROUP_ID	NAME	IMPORTANCE	REQUEST_MAX_MEMORY_GRANULARITY_PERCENT	REQUEST_MAX_CPU_TIME_SEC	REQUEST_MEMORY_GRANULARITY_TIME_OUT_SEC	MAX_DOP	GROUP_MAX_REQUESTS_POOL_ID	POOL_ID	EXTERNAL_POOL_ID
1	internal	Medium	25	0	0	0	0	1	2
2	default	Medium	25	0	0	0	0	2	2
256	ds_wg	Medium	25	0	0	0	0	2	256

2. Use the new catalog view, [sys.resource\\_governor\\_external\\_resource\\_pools \(Transact-SQL\)](#), to view all external resource pools.

```
SELECT * FROM sys.resource_governor_external_resource_pools;
```

### Sample results

EXTERNAL_POOL_ID	NAME	MAX_CPU_PERCENT	MAX_MEMORY_PERCENT	MAX_PROCESSES	VERSION
2	default	100	20	0	2
256	ds_ep	100	40	0	1

For more information, see [Resource Governor Catalog Views \(Transact-SQL\)](#).

3. Run the following statement to return information about the computer resources that are affinitized to the external resource pool, if applicable:

```
SELECT * FROM sys.resource_governor_external_resource_pool_affinity;
```

In this case, because the pools were created with an affinity of AUTO, no information is displayed. For more information, see [sys.dm\\_resource\\_governor\\_resource\\_pool\\_affinity \(Transact-SQL\)](#).

## See also

For more information about managing server resources, see:

- [Resource Governor](#)
- [Resource Governor Related Dynamic Management Views \(Transact-SQL\)](#)

For an overview of resource governance for machine learning, see:

- [Resource Governance for Machine Learning Services](#)

# Extended events for SQL Server Machine Learning Services

4/12/2018 • 5 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server provides a set of extended events to use in troubleshooting operations related to the SQL Server Trusted Launchpad, as well as Python or R jobs sent to SQL Server.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## SQL Server events for machine learning

To view a list of events related to SQL Server, run the following query from SQL Server Management Studio.

```
SELECT o.name AS event_name, o.description
FROM sys.dm_xe_objects o
JOIN sys.dm_xe_packages p
ON o.package_guid = p.guid
WHERE o.object_type = 'event'
AND p.name = 'SQLSatellite';
```

For general information about using extended events, see [Extended Events Tools](#).

### TIP

For extended event generated by SQL Server, try the new [SSMS XEvent profiler](#). This new feature in Management Studio displays a live viewer for extended events, and is less intrusive to the SQL Server than a similar Profiler trace.

## Additional events specific to machine learning components

Additional extended events are available for components that are related to and used by SQL Server Machine Learning Services, such as the SQL Server Trusted Launchpad, and BXLServer, the satellite process that starts the R runtime. These additional extended events are fired from the external processes, and thus must be captured using an external utility.

For more information about how to do this, see the section, [Collecting events from external processes](#).

## Table of extended events

EVENT	DESCRIPTION	NOTES
connection_accept	Occurs when a new connection is accepted. This event serves to log all connection attempts.	
failed_launching	Launching failed.	Indicates an error.
satellite_abort_connection	Abort connection record	

EVENT	DESCRIPTION	NOTES
satellite_abort_received	Fires when an abort message is received over a satellite connection.	
satellite_abort_sent	Fires when an abort message is sent over satellite connection.	
satellite_authentication_completion	Fires when authentication completes for a connection over TCP or Namedpipe.	
satellite_authorization_completion	Fires when authorization completes for a connection over TCP or Namedpipe.	
satellite_cleanup	Fires when satellite calls cleanup.	Fired only from external process. See instructions on collecting events from external processes.
satellite_data_chunk_sent	Fires when the satellite connection finishes sending a single data chunk.	The event reports the number of rows sent, the number of columns, the number of SNI packets usedm and time elapsed in milliseconds while sending the chunk. The information can help you understand how much time is spent passing different types of data, and how many packets are used.
satellite_data_receive_completion	Fires when all the required data by a query is received over the satellite connection.	Fired only from external process. See instructions on collecting events from external processes.
satellite_data_send_completion	Fires when all required data for a session is sent over the satellite connection.	
satellite_data_send_start	Fires when data transmission starts.	Data transmission starts just before the first data chunk is sent.
satellite_error	Used for tracing sql satellite error	
satellite_invalid_sized_message	Message's size is not valid	
satellite_message_coalesced	Used for tracing message coalescing at networking layer	
satellite_message_ring_buffer_record	message ring buffer record	
satellite_message_summary	summary information about messaging	
satellite_message_version_mismatch	Message's version field is not matched	
satellite.messaging	Used for tracing messaging event (bind, unbind etc)	
satellite_partial_message	Used for tracing partial message at networking layer	

EVENT	DESCRIPTION	NOTES
satellite_schema_received	Fires when schema message is received and read by SQL.	
satellite_schema_sent	Fires when schema message is sent by the satellite.	Fired only from external process. See instructions on collecting events from external processes.
satellite_service_start_posted	Fires when service start message is posted to launchpad.	This tells Launchpad to start the external process, and contains an ID for the new session.
satellite_unexpected_message_received	Fires when an unexpected message is received.	Indicates an error.
stack_trace	Occurs when a memory dump of the process is requested.	Indicates an error.
trace_event	Used for tracing purposes	These events can contain SQL Server, Launchpad, and external process trace messages. This includes output to stdout and stderr from R.
launchpad_launch_start	Fires when launchpad starts launching a satellite.	Fired only from Launchpad. See instructions on collecting events from launchpad.exe.
launchpad_resume_sent	Fires when launchpad has launched the satellite and sent a resume message to SQL Server.	Fired only from Launchpad. See instructions on collecting events from launchpad.exe.
satellite_data_chunk_sent	Fires when the satellite connection finishes sending a single data chunk.	Contains information about the number of columns, number of rows, number of packets, and time elapsed sending the chunk.
satellite_sessionId_mismatch	Message's session id is not expected	

## Collecting events from external processes

SQL Server Machine Learning Services starts some services that run outside of the SQL Server process. To capture events related to these external processes, you must create an events trace configuration file and place the file in the same directory as the executable for the process.

- **SQL Server Trusted Launchpad**

To capture events related to the Launchpad, place the `.config` file in the Binn directory for the SQL Server instance. In a default installation, this would be:

```
C:\Program Files\Microsoft SQL Server\MSSQL_version_number.MSSQLSERVER\MSSQL\Binn\
```

- **BXLServer** is the satellite process that supports SQL extensibility with external script languages, such as R or Python. A separate instance of BxlServer is launched for each external language instance.

To capture events related to BXLServer, place the `.config` file in the R or Python installation directory. In a default installation, this would be:

**R:**

```
C:\Program Files\Microsoft SQL
Server\MSSQL_version_number.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64
```

### Python:

```
C:\Program Files\Microsoft SQL
Server\MSSQL_version_number.MSSQLSERVER\PYTHON_SERVICES\library\RevoScaleR\rxLibs\x64
```

The configuration file must be named the same as the executable, using the format “[name].xevents.xml”. In other words, the files must be named as follows:

- `Launchpad.xevents.xml`
- `bxlserver.xevents.xml`

The configuration file itself has the following format:

```
\<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
 <event_session name="[session name]" maxMemory="1" dispatchLatency="1" MaxDispatchLatency="2 SECONDS">
 <description owner="you">Xevent for launchpad or bxl server.</description>
 <event package="SQLSatellite" name="[XEvent Name 1]" />
 <event package="SQLSatellite" name="[XEvent Name 2]" />
 <target package="package0" name="event_file">
 <parameter name="filename" value="[SessionName].xel" />
 <parameter name="max_file_size" value="10" />
 <parameter name="max_rollover_files" value="10" />
 </target>
 </event_session>
</event_sessions>
```

- To configure the trace, edit the *session name* placeholder, the placeholder for the filename (`[SessionName].xel`), and the names of the events you want to capture, For example, `[XEvent Name 1]`, `[XEvent Name 2]`.
- Any number of event package tags may appear, and will be collected as long as the name attribute is correct.

### Example: Capturing Launchpad events

The following example shows the definition of an event trace for the Launchpad service:

```
\<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
 <event_session name="sqlsatelliteut" maxMemory="1" dispatchLatency="1" MaxDispatchLatency="2 SECONDS">
 <description owner="hay">Xevent for sql tdd runner.</description>
 <event package="SQLSatellite" name="launchpad_launch_start" />
 <event package="SQLSatellite" name="launchpad_resume_sent" />
 <target package="package0" name="event_file">
 <parameter name="filename" value="launchpad_session.xel" />
 <parameter name="max_file_size" value="10" />
 <parameter name="max_rollover_files" value="10" />
 </target>
 </event_session>
</event_sessions>
```

- Place the `.config` file in the Binn directory for the SQL Server instance.
- This file must be named `Launchpad.xevents.xml`.

### Example: Capturing BXLServer events

The following example shows the definition of an event trace for the BXLServer executable.

```
\<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
 <event_session name="sqlsatelliteut" maxMemory="1" dispatchLatency="1" MaxDispatchLatency="2 SECONDS">
 <description owner="hay">Xevent for sql tdd runner.</description>
 <event package="SQLSatellite" name="satellite_abort_received" />
 <event package="SQLSatellite" name="satellite_authentication_completion" />
 <event package="SQLSatellite" name="satellite_cleanup" />
 <event package="SQLSatellite" name="satellite_data_receive_completion" />
 <event package="SQLSatellite" name="satellite_data_send_completion" />
 <event package="SQLSatellite" name="satellite_data_send_start" />
 <event package="SQLSatellite" name="satellite_schema_sent" />
 <event package="SQLSatellite" name="satellite_unexpected_message_received" />
 <event package="SQLSatellite" name="satellite_data_chunk_sent" />
 <target package="package0" name="event_file">
 <parameter name="filename" value="satellite_session.xel" />
 <parameter name="max_file_size" value="10" />
 <parameter name="max_rollover_files" value="10" />
 </target>
 </event_session>
</event_sessions>
```

- Place the `.config` file in the same directory as the BXLServer executable.
- This file must be named `bxlserver.xevents.xml`.

## See also

[Custom Management Studio Reports for Machine Learning Services](#)

# Extended events for monitoring PREDICT statements

4/11/2018 • 2 minutes to read • [Edit Online](#)

This article describes the extended events provided in SQL Server that you can use to monitor and analyze jobs that use **PREDICT** to perform real-time scoring in SQL Server.

Real-time scoring generates scores from a machine learning model that has been stored in SQL Server. The PREDICT function does not require an external run-time such as R or Python, only a model that has been created using a specific binary format. For more information, see [Realtime scoring](#).

## Prerequisites

For general information about extended events (sometimes called XEvents), and how to track events in a session, see these articles:

- [Extended Events concepts and architecture](#)
- [Set up event capture in SSMS](#)
- [Manage event sessions in the Object Explorer](#)

## Table of extended events

The following extended events are available in all versions of SQL Server that support the [T-SQL PREDICT](#) statement, including SQL Server on Linux, and Azure SQL Database.

The T-SQL PREDICT statement was introduced in SQL Server 2017.

NAME	OBJECT_TYPE	DESCRIPTION
predict_function_completed	event	Builtin execution time breakdown
predict_model_cache_hit	event	Occurs when a model is retrieved from the PREDICT function model cache. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.
predict_model_cache_insert	event	Occurs when a model is insert into the PREDICT function model cache. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.
predict_model_cache_miss	event	Occurs when a model is not found in the PREDICT function model cache. Frequent occurrences of this event could indicate that SQL Server needs more memory. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.

NAME	OBJECT_TYPE	DESCRIPTION
predict_model_cache_remove	event	Occurs when a model is removed from model cache for PREDICT function. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.

## Query for related events

To view a list of all columns returned for these events, run the following query in SQL Server Management Studio:

```
SELECT * FROM sys.dm_xe_object_columns WHERE object_name LIKE `predict%`
```

## Examples

To capture information about performance of a scoring session using PREDICT:

1. Create a new extended event session, using Management Studio or another supported [tool](#).
2. Add the events `predict_function_completed` and `predict_model_cache_hit` to the session.
3. Start the extended event session.
4. Run the query that uses PREDICT.

In the results, review these columns:

- The value for `predict_function_completed` shows how much time the query spent on loading the model and scoring.
- The boolean value for `predict_model_cache_hit` indicates whether the query used a cached model or not.

### Native scoring model cache

In addition to the events specific to PREDICT, you can use the following queries to get more information about the cached model and cache usage:

View the native scoring model cache:

```
SELECT *
FROM sys.dm_os_memory_clerks
WHERE type = 'CACHESTORE_NATIVESCORING';
```

View the objects in the model cache:

```
SELECT *
FROM sys.dm_os_memory_objects
WHERE TYPE = 'MEMOBJ_NATIVESCORING';
```

# DMVs for SQL Server Machine Learning Services

4/12/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The article lists the system catalog views and DMVs that are related to machine learning in SQL Server.

For information about extended events, see [Extended events for machine learning](#).

## TIP

Use built-in reports to monitor machine learning sessions and package utilization. For more information, see [Monitor machine learning using custom reports in Management Studio](#).

## System configuration and system resources

You can monitor and analyze the resources used by external scripts by using SQL Server system catalog views and DMVs.

- [sys.dm\\_exec\\_sessions](#)

Returns information for both user connections and system sessions. You can identify the system sessions by looking at the `session_id` column; values greater than or equal to 51 are user connections and values lower than 51 are system processes.

- [sys.dm\\_os\\_performance\\_counters \(Transact-SQL\)](#)

Returns a row for each system performance counter being used by the server. You can use this information to see how many scripts ran, which scripts were run using which authentication mode, or how many R calls were issued on the instance overall.

This example gets just the counters related to R script:

```
SELECT * from sys.dm_os_performance_counters WHERE object_name LIKE '%External Scripts%'
```

The following counters are reported by this DMV for external scripts per instance:

- **Total Executions:** Number of external processes started by local or remote calls
- **Parallel Executions:** Number of times that a script included the `@parallel` specification and that SQL Server was able to generate and use a parallel query plan
- **Streaming Executions:** Number of times that the streaming feature has been invoked
- **SQL CC Executions:** Number of external scripts run where the call was instantiated remotely and SQL Server was used as the compute context
- **Implied Auth. Logins:** Number of times that an ODBC loopback call was made using implied authentication; that is, the SQL Server executed the call on behalf of the user sending the script request
- **Total Execution Time (ms):** Time elapsed between the call and completion of call
- **Execution Errors:** Number of times scripts reported errors. This count does not include R errors.
- [sys.dm\\_external\\_script\\_requests](#)

This DMV reports a single row for each worker account that is currently running an external script. Note

that this worker account is different from the credentials of the person sending the script. If a single Windows user sends multiple script requests, only one worker account would be assigned to handle all requests from that user. If a different Windows user logs in to run an external script, the request would be handled by a separate worker account.

This DMV does not return any results if no scripts are currently being executed; thus, it is most useful for monitoring long-running scripts. It returns these values:

- **external\_script\_request\_id**: A GUID, which is also used as the temporary name of the working directory used to store scripts and intermediate results
- **language**: A value such as `R` that denotes the language of the external script
- **degree\_of\_parallelism**: An integer indicating the number of parallel processes that were used
- **external\_user\_name**: A Launchpad worker account, such as `SQLRUser01`
- [sys.dm\\_external\\_script\\_execution\\_stats \(Transact-SQL\)](#)

This DMV is provided for internal monitoring (telemetry) to track how many external script calls are made on an instance. The telemetry service starts when SQL Server does and increments a disk-based counter each time a specific machine learning function is called.

The counter is incremented per call to a specific tracked function. For example, if `rxLinMod` is called and run in parallel, the counter is incremented by 1.

Generally speaking, performance counters are valid only as long as the process that generated them is active. Therefore, a query on a DMV cannot show detailed data for services that have stopped running. For example, if the Launchpad creates multiple parallel R jobs and yet they are very quickly executed and then cleaned up by the Windows job object, a DMV might not show any data.

However, the counters tracked by this DMV are kept running, and state for `dm_external_script_execution` counter is preserved by using writes to disk, even if the instance is shut down.

For more information about system performance counters used by SQL Server, see [Use SQL Server Objects](#).

## Resource Governor views

In editions that support Resource Governor, creating external resource pools for R or Python workloads can be an effective way to track and manage resources.

- [sys.resource\\_governor\\_resource\\_pools](#)

Returns information about the current resource pool state, the current configuration of resource pools, and resource pool statistics.

### IMPORTANT

You must modify resource pools that apply to other server services before you can allocate additional resources to R Services.

- [sys.resource\\_governor\\_external\\_resource\\_pools](#)

A new catalog view that shows the current configuration values for external resource pools. In Enterprise Edition, you can configure additional external resource pools: for example, you might decide to handle resources for R jobs running in SQL Server separately from those that originate from a remote client.

#### **NOTE**

In Standard Edition, all external script jobs execute within the same external default resource pool.

- [sys.resource\\_governor\\_workload\\_groups](#)

Returns workload group statistics and the current configuration of the workload group. This view can be joined with `sys.dm_resource_governor_resource_pools` to get the resource pool name. For external scripts, a new column has been added that shows the id of the external pool associated with the workload group.

- [sys.dm\\_resource\\_governor\\_external\\_resource\\_pool\\_affinity](#)

A new system catalog view that lets you see the processors and resources that are affinitized to a particular resource pool.

Returns one row per scheduler in SQL Server where each scheduler is mapped to an individual processor. Use this view to monitor the condition of a scheduler or to identify runaway tasks.

Under the default configuration, workload pools are automatically assigned to processors and therefore there are no affinity values to return.

The affinity schedule maps the resource pool to the SQL Server schedules identified by the given IDs. These IDs map to the values in the `scheduler_id` column in `sys.dm_osSchedulers`.

#### **NOTE**

Although the ability to configure and customize resource pools is available only in Enterprise and Developer editions, the default pools as well as the DMVs are available in all editions. Therefore, you can use these DMVs in Standard Edition to determine resource caps for your external script jobs.

For general information about monitoring SQL Server instances, see [Catalog Views](#) and [Resource Governor Related Dynamic Management Views](#).

## Monitoring script execution

R and Python scripts that run in SQL Server are started by the SQL Server Trusted Launchpad interface. However, the Launchpad is not resource governed or monitored separately, as it is assumed to be a secure service provided by Microsoft that manages resources appropriately.

Individual scripts that run under the Launchpad service are managed using the [Windows job object](#). A job object allows groups of processes to be managed as a unit. Each job object is hierarchical and controls the attributes of all processes associated with it. Operations performed on a job object affect all processes associated with the job object.

Thus, if you need to terminate one job associated with an object, be aware that all related processes will also be terminated. If you are running an R script that is assigned to a Windows job object and that script runs a related ODBC job which must be terminated, the parent R script process will be terminated as well.

If you start an external script that uses parallel processing, a single Windows job object manages all parallel child processes.

To determine if a process is running in a job, use the `IsProcessInJob` function.

## Next steps

[Managing and monitoring machine learning solutions](#)



# Using R code profiling functions

4/12/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (Windows only) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

In addition to using SQL Server resources and tools to monitor R script execution, you can use performance tools provided by other R packages to get more information about internal function calls. This topic provides a list of some basic resources to get you started. For expert guidance, we recommend the chapter on [Performance](#) in the book ""Advanced R"" by Hadley Wickham.

## Using RPROF

*rprof* is a function included in the base package **utils**, which is loaded by default. One advantage of *rprof* is that it performs sampling, thus lessening the performance load from monitoring.

To use R profiling in your code, you call this function and specify its parameters, including the name of the location of the log file that will be written. See the help for *rprof* for details.

In general, the *rprof* function works by writing out the call stack to a file, at specified intervals. You can then use the *summaryRprof* function to process the output file.

Profiling can be turned on and off in your code. To turn profiling on, suspend profiling, and then restart it, you would use a sequence of calls to *rprof*:

1. Specify profiling output file.

```
varOutputFile <- "C:/TEMP/run001.log"
Rprof(varOutputFile)
```

2. Turn off profiling

```
Rprof(NULL)
```

3. Restart profiling

```
Rprof(append=TRUE)
```

### NOTE

Using this function requires that Windows Perl be installed on the computer where code is run. Therefore, we recommend that you profile code during development in an R environment, and then deploy the debugged code to SQL Server.

## R System Functions

The R language includes many base package functions for returning the contents of system variables.

For example, as part of your R code, you might use `Sys.timezone` to get the current time zone, or `Sys.Time` to get the system time from R.

To get information about individual R system functions, type the function name as the argument to the R `help()` function from an R command prompt.

```
help("Sys.time")
```

## Debugging and Profiling in R

The documentation for Microsoft R Open, which is installed by default, includes a manual on developing extensions for the R language that discusses profiling and debugging in detail.

The chapter is also available online: <https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Debugging>

### Location of R help files

C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R\_SERVICES\doc\manual

# Monitor Machine Learning Services using custom reports in Management Studio

4/12/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

To make it easier to manage instance used for machine learning, the product team has provided a number of sample custom reports that you can add to SQL Server Management Studio. In these reports, you can view details such as:

- Active R or Python sessions
- Configuration settings for the instance
- Execution statistics for machine learning jobs
- Extended events for R Services
- R or Python packages installed on the current instance

This article explains how to install and use the custom reports provided specifically for machine learning.

For a general introduction to reports in Management Studio, see [Custom reports in Management Studio](#).

## How to install the reports

The reports are designed using SQL Server Reporting Services, but can be used directly from SQL Server Management Studio, even if Reporting Services is not installed on your instance.

To use these reports:

- Download the RDL files from the GitHub repository for SQL Server product samples.
- Add the files to the custom reports folder used by SQL Server Management Studio.
- Open the reports in SQL Server Management Studio.

### Step 1. Download the reports

1. Open the GitHub repository that contains [SQL Server product samples](#), and download the sample reports.

- [SSMS custom reports](#)

#### NOTE

The reports can be used with either SQL Server 2017 Machine Learning Services, or SQL Server 2016 R Services.

2. To download the samples, you can also log into GitHub and make a local fork of the samples.

### Step 2. Copy the reports to Management Studio

1. Locate the custom reports folder used by SQL Server Management Studio. By default, custom reports are stored in this folder:

C:\Users\user\_name\Documents\SQL Server Management Studio\Custom Reports

However, you can specify a different folder, or create subfolders.

2. Copy the \*.RDL files to the custom reports folder.

### Step 3. Run the reports

1. In Management Studio, right-click the **Databases** node for the instance where you want to run the reports.
2. Click **Reports**, and then click **Custom Reports**.
3. In the **Open File** dialog box, locate the custom reports folder.
4. Select one of the RDL files you downloaded, and then click **Open**.

#### IMPORTANT

On some computers, such as those with display devices with high DPI or greater than 1080p resolution, or in some remote desktop sessions, these reports cannot be used. There is a bug in the report viewer control in SSMS that crashes the report.

## Report list

The product samples repository in GitHub currently includes the following reports:

- **R Services - Active Sessions**

Use this report to view the users who are currently connected to the SQL Server instance and running machine learning jobs.

- **R Services - Configuration**

Use this report to view the configuration of the external script runtime and related services. The report will indicate whether a restart is required, and will check for required network protocols.

Implied authentication is required for machine learning tasks that run in SQL Server as a compute context. To verify that implied authentication is configured, the report verifies whether a database login exists for the group SQLRUserGroup.

- **R Services - Configure Instance**

This report is intended to help you configure machine learning. You can also run this report to fix configuration errors found in the preceding report.

- **R Services - Execution Statistics**

Use this report to view execution statistics for machine learning jobs. For example, you can get the total number of R scripts that were executed, the number of parallel executions, and the most frequently used RevoScaleR functions. Click **View SQL Script** to get the complete T-SQL code behind this report.

Currently the report monitors only statistics for RevoScaleR package functions.

- **R Services - Extended Events**

Use this report to view a list of the extended events that are available for monitoring tasks related to external script runtimes. Click **View SQL Script** to get the complete T-SQL code behind this report.

- **R Services - Packages**

Use this report to view a list of the R or Python packages installed on the SQL Server instance.

- **R Services - Resource Usage**

Use this report to view consumption of CPU, memory, and I/O resources by external script execution. You can also view the memory setting of external resource pools.

## See also

[Monitoring services](#)

[Extended events for R Services](#)

# API reference for SQL Server Machine Learning Services

5/29/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article provides links to the reference documentation for machine learning APIs used by SQL Server.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

For the most part, SQL Server consumes the same R and Python libraries that are provided in Microsoft R Server and Microsoft Machine Learning Server.

## NOTE

Documentation for all APIs is derived from source code and has not been edited. If you see errors, please add a comment in the API reference documentation.

## R

- [RevoScaleR](#)

Scalable algorithms that support remote compute contexts and multiple data sources.

- [MicrosoftML](#)

Fast, scalable machine learning algorithms and transforms for R. Requires RevoScaleR.

- [olapR](#)

Reads the schema of OLAP data sources and executes MDX queries.

- [sqlrutils](#)

Helper functions for generating a well-formed stored procedure from R code.

- [mrsdeploy](#)

Functions for establishing a remote session in a console application and for publishing and managing a web service that uses R or Python code.

## Python

- [revoscalepy](#)

Python equivalent of the RevoScaleR package for the R language. Supports the same compute contexts and data sources.

- [Microsoftml for Python](#)

Python equivalent of the MicrosoftML package for R. Supports the same compute contexts and data sources, and includes fast, scalable algorithms and transformations from Microsoft.

## Related APIs

- [RevoPEMAR function reference](#)

Supports development of parallel algorithms

- [RevoUtils](#)

Utility functions for use with RevoScaleR environments

## Other

"How-to" topics and summaries specific to use of these R or Python APIs in SQL Server can be found here:

- [ScaleR functions for working with SQL Server](#)
- [Generate a stored procedure using sqlrutils](#)
- [Read MDX Data into R using olapR](#)

# Using the MicrosoftML package with SQL Server

7/11/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The **MicrosoftML** package that is provided with Microsoft R Server and SQL Server 2017 includes multiple machine learning algorithms. These APIs were developed by Microsoft for internal machine learning applications, and have been refined over the years to support high performance on big data, using multicore processing and fast data streaming. MicrosoftML also includes numerous transformations for text and image processing.

In SQL Server 2017 CTP 2.0, support was added for the Python language. The **microsoftml** package for Python contains functions equivalent to those in the MicrosoftML package for R.

## • MicrosoftML for R

Introduction and package reference: [MicrosoftML: machine learning R algorithms](#)

Because R is case-sensitive, make sure that you reference the name correctly when loading the package.

## • microsoftml for Python

Introduction and package reference: [microsoftml \(Function library for Python\)](#).

## What's in MicrosoftML

MicrosoftML contains a variety of machine learning algorithms and transformations that have been optimized for performance.

### Machine learning algorithms

- Linear models: `rxFastLinear` is a linear learner based on stochastic dual coordinate ascent that can be used for binary classification or regression. The model supports L1 and L2 regularization.
- Decision tree and decision forest models: `rxFastTree` is a boosted decision tree algorithm originally known as FastRank, which was developed for use in Bing. It is one of the fastest and most popular learners. Supports binary classification and regression.

`rxFastForest` is a logistic regression model based on the random forest method. It is similar to the `rxLogit` function in RevoScaleR, but supports L1 and L2 regularization. Supports binary classification and regression.

- Logistic regression: `rxLogisticRegression` is a logistic regression model similar to the `rxLogit` function in RevoScaleR, with additional support for L1 and L2 regularization. Supports binary or multiclass classification.
- Neural networks: The `rxNeuralNet` function supports binary classification, multiclass classification, and regression using neural networks. Customizable and supports convoluted networks with GPU acceleration, using a single GPU.
- Anomaly detection. The `rxOneClassSvm` function is based on the SVM method but is optimized for binary classification in unbalanced data sets.

### Transformation functions

MicrosoftML includes numerous specialized functions that are useful for transforming data and extracting

features.

- Text processing capabilities include the `featurizeText` and `getSentiment` functions. You can count n-grams, detect the language used, or perform text normalization. You can also perform common text cleaning operations such as stopword removal, or generate hashed or count-based features from text.
- Feature selection and feature transformation functions, such as `selectFeatures` or `getSentiment`, analyze data and create features that are most useful for modeling.
- Work with categorical variables by using such as `categorical` or `categoricalHash`, which convert categorical values into indexed arrays for better performance.
- Functions specific to image processing and analytics, such as `extractPixels` or `featurizeImage`, let you get the most information about of images and process images faster.

For more information, see [MicrosoftML functions](#).

## How to use MicrosoftML

This section describes how to locate and load the package in your R and Python code.

- The MicrosoftML package for R is installed by default with Microsoft R Server 9.1.0 and in SQL Server 2017.

It is also available for use with SQL Server 2016, if you upgrade the R components for the instance, by using the Microsoft R Server installer as described here: [Upgrade an instance of SQL Server using binding](#)

- The **microsoftml** package for Python is installed by default with SQL Server 2017

To use this package, we recommend that you upgrade to Release Candidate 2 or later. An early version was released with RC1 but the library has undergone considerable revision, including changes to function names.

However, for both R and Python, the package is not loaded by default; thus, you must explicitly load the package as part of your code to use its functions.

### Calling MicrosoftML functions from R in SQL Server

In your R code, load the MicrosoftML package and call its functions, like any other package.

```
library(microsoftml);
library(RevoScaleR);
logisticRegression(args);
```

### Notes

- The MicrosoftML package is fully integrated with the data processing pipeline provided by the RevoScaleR package. Thus, you can use the MicrosoftML package in any Windows-based compute context, including an instance of SQL Server that has machine learning extensions enabled.

However, MicrosoftML requires a reference to RevoScaleR and its functions in order to use remote compute contexts.

### Calling microsoftml functions from Python in SQL Server

To call functions from the package, in your Python code, import the **microsoftml** package, and import **revoscalepy** if you need to use remote compute contexts or related connectivity or data source objects. Then, reference the individual functions you need.

```
from microsoftml.modules.logistic_regression.rx_logistic_regression import rx_logistic_regression
from revoscalepy.functions.RxSummary import rx_summary
from revoscalepy.etl.RxImport import rx_import_datasource
```

## Notes

- In SQL Server 2017, **microsoftml** is a Python35-compatible module.
- The functions in **microsoftml** are integrated with the compute contexts and data sources that are supported by **revoscalepy**. Thus, you can use the **microsoftml** Python package to create and score from models in any Windows-based compute context, including an instance of SQL Server that has machine learning extensions. enabled.

However, **microsoftml** for Python requires a reference to **revoscalepy** and its functions in order to use remote compute contexts.

For more information about revoscalepy, see:

- [What is revoscalepy](#)
- [revoscalepy function library](#)

# RevoScaleR

4/12/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

RevoScaleR is a package of machine learning functions, provided by Microsoft, that supports data science at scale.

- Functions support data import, data transformation, summarization, visualization, and analysis.
- *At scale* means that operations can be performed against very large datasets, in parallel, and on distributed file systems. Algorithms can operate over datasets that do not fit in memory, by using chunking and by reassembling results when operations are complete.
- RevoScaleR provides many improvements over open source R functions. There are RevoScaleR functions corresponding to many of the most popular base R functions. RevoScaleR functions are denoted with an **rx** or **Rx** prefix to make them easy to identify.
- RevoScaleR serves as a platform for distributed data science. For example, you can use the RevoScaleR compute contexts and transformations with the state-of-the-art algorithms in [MicrosoftML](#). You can also use [rxExec](#) to run base R functions in parallel.

For examples of RevoScaleR in action, see these blogs:

- [Build and deploy a predictive model using R and SQL Server](#)
- [One million predictions per second with Machine Learning Server](#)
- [Predicting taxi tips using MicrosoftML](#)
- [Performance optimization with rxExec](#)

## How to get RevoScaleR

The RevoScaleR package for R is installed for free in Microsoft R Client. If you have Machine Learning Server or are using R in SQL Server, RevoScaleR is included by default.

If you are using Python, the [revoscalepy](#) package provides equivalent functionality.

### IMPORTANT

The RevoScaleR package cannot be downloaded or used independently of the products and services that provide it.

## Use RevoScaleR in SQL Server

These tutorials and samples demonstrate how to use RevoScaleR functions to get data from SQL Server, build models, and save models to a database for scoring.

- [Learn to use compute contexts](#)
- [R for SQL developers: Train and operationalize a model](#)
- [Microsoft product samples on GitHub](#)

## Learn more about RevoScaleR

These tutorials demonstrate the use of RevoScaleR in other compute contexts supported by [Machine Learning Server](#), including Hadoop.

- [What is RevoScaleR?](#)
- [Explore RevoScaleR in 25 functions](#)
- [RevoScaleR package reference](#)

# RevoScaleR functions for working with SQL Server data

4/12/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic provides an overview of the main functions provided in RevoScaleR for working with SQL Server data.

For a complete list of ScaleR functions and how to use them, see the [Microsoft R Server](#) reference.

## Create SQL Server data sources

The following functions let you define a SQL Server data source. A data source object is a container that specifies a connection string together with the set of data that you want, defined either as a table, view, or query. Stored procedure calls are not supported.

- [RxSqlServerData](#) - Define a SQL Server data source object.
- [RxOdbcData](#) - Create data objects for other ODBC databases.

## Perform DDL statements

You can execute DDL statements from R, if you have the necessary permissions on the instance and database. The following functions use ODBC calls to execute DDL statements or retrieve the database schema.

- `rxSqlServerTableExists` and [rxSqlServerDropTable](#) - Drop a SQL Server table, or check for the existence of a database table or object
- [rxExecuteSQLDDL](#) - Execute a Data Definition Language (DDL) command that defines or manipulates database objects. This function cannot return data, and is used only to retrieve or modify the object schema or metadata.

## Define or manage compute contexts

The following functions let you define a new compute context, switch compute contexts, or identify the current compute context.

- [RxComputeContext](#) - Create a compute context.
- [rxInSqlServer](#) - Generate a SQL Server compute context that lets **ScaleR** functions run in SQL Server R Services. This compute context is currently supported only for SQL Server instances on Windows.
- `rxGetComputeContext` and [rxSetComputeContext](#) - Get or set the active compute context.

## Move data and transform data

After you have created a data source object, you can use the object to load data into it, transform data, or write new data to the specified destination. Depending on the size of the data in the source, you can also define the batch size as part of the data source and move data in chunks.

- [rxOpen-methods](#) - Check whether a data source is available, open or close a data source, read data from a source, write data to the target, and close a data source

- [rxImport](#) - Move data from a data source into file storage or into a data frame.
- [rxDataStep](#) - Transform data while moving it between data sources.

The following functions can be used to create a local data store in the XDF format. This file can be useful when working with more data than can be transferred from the database in one batch, or more data than can fit in memory.

For example, if you regularly move large amounts of data from a database to a local workstation, rather than query the database repeatedly for each R operation, you can use the XDF file as a kind of cache to save the data locally and then work with it in your R workspace.

- [RxXdfData](#) - Create an XDF data object
- [rxReadXdf](#) - Reads data from an XDF file into a data frame

For more information about working with these functions, including using data sources other than SQL Server, see [Howto guides for data analysis in Microsoft R](#).

# Generate an R stored procedure for R Code using the `sqlrutils` package

4/12/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The **sqlrutils** package provides a mechanism for R users to put their R scripts into a T-SQL stored procedure, register that stored procedure with a database, and run the stored procedure from an R development environment.

By converting your R code to run within a single stored procedure, you can make more effective use of SQL Server R Services, which requires that R script be embedded as a parameter to `sp_execute_external_script`. The **sqlrutils** package helps you build this embedded R script and set related parameters appropriately.

The **sqlrutils** package performs these tasks:

- Saves the generated T-SQL script as a string inside an R data structure
- Optionally, generates a .sql file for the T-SQL script, which you can edit or run to create a stored procedure
- Registers the newly created stored procedure with the SQL Server instance from your R development environment

You can also execute the stored procedure from an R environment, by passing well-formed parameters and processing the results. Or, you can use the stored procedure from SQL Server to support common database integration scenarios such as ETL, model training, and high-volume scoring.

## NOTE

If you intend to run the stored procedure from an R environment by calling the `executeStoredProcedure` function, you must use an ODBC 3.8 provider, such as ODBC Driver 13 for SQL Server.

## Functions provided in `sqlrutils`

The following list provides an overview of the functions that you can call from the **sqlrutils** package to develop a stored procedure for use in SQL Server R Services. For details of the parameters for each method or function, see the R help for the package:

```
help(package="sqlrutils")
```

## Define stored procedure parameters and inputs

- `InputData` . Defines the source of data in SQL Server that will be used in the R data frame. You specify the name of the data.frame in which to store the input data, and a query to get the data, or a default value. Only simple SELECT queries are supported.
- `InputParameter` . Defines a single input parameter that will be embedded in the T-SQL script. You must provide the name of the parameter and its R data type.
- `OutputData` . Generates an intermediate data object that is needed if your R function returns a list that contains a data.frame. The `OutputData` object is used to store the name of a single data.frame obtained

from the list.

- `OutputParameter`. Generates an intermediate data object that is needed if your R function returns a list. The `OutputParameter` object stores the name and data type of a single member of the list, assuming that member is **not** a data frame.

## Generate and register the stored procedure

- `StoredProcedure` is the main constructor used to build the stored procedure. This constructor generates a *SQL Server Stored Procedure* object, and optionally creates a text file containing a query that can be used to generate the stored procedure using a T-SQL command. Optionally, the `StoredProcedure` function can also register the stored procedure with the specified instance and database.
  - Use the `func` argument to specify a valid R function. All the variables that the function uses must be defined either inside the function or be provided as input parameters. These parameters can include a maximum of one data frame.
  - The R function must return either a data frame, a named list, or a NULL. If the function returns a list, the list can contain a maximum of one data.frame.
  - Use the argument `spName` to specify the name of the stored procedure you want to create.
  - You can pass in optional input and output parameters, using the objects created by these helper functions: `setInputData`, `setInputParameter`, and `setOutputParameter`.
  - Optionally, use `filePath` to provide the path and name of a .sql file to create. You can run this file on the SQL Server instance to generate the stored procedure using T-SQL.
  - To define the server and database where the stored procedure will be saved, use the arguments `dbName` and `connectionString`.
  - To get a list of the `InputData` and `InputParameter` objects that were used to create a specific `StoredProcedure` object, call `getInputParameters`.
  - To register the stored procedure with the specified database, use `register.StoredProcedure`.

The stored procedure object typically does not have any data or values associated with it, unless a default value was specified. Data is not retrieved until the stored procedure is executed.

## Specify inputs and execute

- Use `setInputDataQuery` to assign a query to an `InputParameter` object. For example, if you have created a stored procedure object in R, you can use `setInputDataQuery` to pass arguments to the `StoredProcedure` function in order to execute the stored procedure with the desired inputs.
- Use `setInputValue` to assign specific values to a parameter stored as an `InputParameter` object. You then pass the parameter object and its value assignment to the `StoredProcedure` function to execute the stored procedure with the set values.
- Use `execute.StoredProcedure` to execute a stored procedure defined as an `StoredProcedure` object. Call this function only when executing a stored procedure from R code. Do not use it when running the stored procedure from SQL Server using T-SQL.

### NOTE

The `execute.StoredProcedure` function requires an ODBC 3.8 provider, such as ODBC Driver 13 for SQL Server.

## See Also

[How to Create a Stored Procedure using sqlrutils](#)

# How to create MDX queries in R using olapR

4/12/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The `olapR` package supports MDX queries against cubes hosted in SQL Server Analysis Services. You can build a query against an existing cube, explore dimensions and other cube objects, and paste in existing MDX queries to retrieve data.

This article describes the two main uses of the `olapR` package:

- [Build an MDX query from R, using the constructors provided in the olapR package](#)
- [Execute an existing, valid MDX query using olapR and an OLAP provider](#)

The following operations are not supported:

- DAX queries against a tabular model
- Creation of new OLAP objects
- Writeback to partitions, including measures or sums

## Build an MDX query from R

1. Define a connection string that specifies the OLAP data source (SSAS instance), and the MSOLAP provider.
2. Use the function `OlapConnection(connectionString)` to create a handle for the MDX query and pass the connection string.
3. Use the `Query()` constructor to instantiate a query object.
4. Use the following helper functions to provide more details about the dimensions and measures to include in the MDX query:
  - `cube()` Specify the name of the SSAS database. If connecting to a named instance, provide the machine name and instance name.
  - `columns()` Provide the names of the measures to use in the **ON COLUMNS** argument.
  - `rows()` Provide the names of the measures to use in the **ON ROWS** argument.
  - `slicers()` Specify a field or members to use as a slicer. A slicer is like a filter that is applied to all MDX query data.
  - `axis()` Specify the name of an additional axis to use in the query.

An OLAP cube can contain up to 128 query axes. Generally, the first four axes are referred to as **Columns, Rows, Pages, and Chapters**.

If your query is relatively simple, you can use the functions `columns`, `rows`, etc. to build your query. However, you can also use the `axis()` function with a non-zero index value to build an MDX query with many qualifiers, or to add extra dimensions as qualifiers.

5. Pass the handle, and the completed MDX query, into one of the following functions, depending on the shape of the results:
  - `executeMD` Returns a multi-dimensional array

- `execute2D` Returns a two-dimensional (tabular) data frame

## Execute a valid MDX query from R

1. Define a connection string that specifies the OLAP data source (SSAS instance), and the MSOLAP provider.
2. Use the function `OlapConnection(connectionString)` to create a handle for the MDX query and pass the connection string.
3. Define an R variable to store the text of the MDX query.
4. Pass the handle and the variable containing the MDX query into the functions `executeMD` or `execute2D`, depending on the shape of the results.
  - `executeMD` Returns a multi-dimensional array
  - `execute2D` Returns a two-dimensional (tabular) data frame

## Examples

The following examples are based on the AdventureWorks data mart and cube project, because that project is widely available, in multiple versions, including backup files that can easily be restored to Analysis Services. If you don't have an existing cube, get a sample cube using either of these options:

- Create the cube that is used in these examples by following the Analysis Services tutorial up to Lesson 4: [Creating an OLAP cube](#)
- Download an existing cube as a backup, and restore it to an instance of Analysis Services. For example, this site provides a fully processed cube in zipped format: [Adventure Works Multidimensional Model SQL 2014](#). Extract the file, and then restore it to your SSAS instance. For more information, see [Backup and restore](#), or [Restore-ASDatabase Cmdlet](#).

### 1. Basic MDX with slicer

This MDX query selects the *measures* for count and amount of Internet sales count and sales amount, and places them on the Column axis. It adds a member of the SalesTerritory dimension as a *slicer*, to filter the query so that only the sales from Australia are used in calculations.

```
SELECT {[Measures].[Internet Sales Count], [Measures].[InternetSales-Sales Amount]} ON COLUMNS,
 {[Product].[Product Line].[Product Line].MEMBERS} ON ROWS
 FROM [Analysis Services Tutorial]
 WHERE [Sales Territory].[Sales Territory Country].[Australia]
```

- On columns, you can specify multiple measures as elements of a comma-separated string.
- The Row axis uses all possible values (all MEMBERS) of the "Product Line" dimension.
- This query would return a table with three columns, containing a *rollup* summary of Internet sales from all countries.
- The WHERE clause specifies the *slicer axis*. In this example, the slicer uses a member of the **SalesTerritory** dimension to filter the query so that only the sales from Australia are used in calculations.

**To build this query using the functions provided in olapR**

```

cnnstr <- "Data Source=localhost; Provider=MSOLAP;"

ocs <- OlapConnection(cnnstr)

qry <- Query()

cube(qry) <- "[Analysis Services Tutorial]"

columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].[Internet Sales-Sales Amount]")

rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")

slicers(qry) <- c("[Sales Territory].[Sales Territory Country].[Australia]")

result1 <- executeMD(ocs, qry)

```

For a named instance, be sure to escape any characters that could be considered control characters in R. For example, the following connection string references an instance OLAP01, on a server named ContosoHQ:

```
cnnstr <- "Data Source=ContosoHQ\\OLAP01; Provider=MSOLAP;"
```

#### To run this query as a predefined MDX string

```

cnnstr <- "Data Source=localhost; Provider=MSOLAP;"

ocs <- OlapConnection(cnnstr)

mdx <- "SELECT {[Measures].[Internet Sales Count], [Measures].[InternetSales-Sales Amount]} ON COLUMNS,

{[Product].[Product Line].[Product Line].MEMBERS} ON ROWS FROM [Analysis Services Tutorial] WHERE [Sales

Territory].[Sales Territory Country].[Australia]"

result2 <- execute2D(ocs, mdx)

```

If you define a query by using the MDX builder in SQL Server Management Studio and then save the MDX string, it will number the axes starting at 0, as shown here:

```

SELECT {[Measures].[Internet Sales Count], [Measures].[Internet Sales-Sales Amount]} ON AXIS(0),

{[Product].[Product Line].[Product Line].MEMBERS} ON AXIS(1)

FROM [Analysis Services Tutorial]

WHERE [Sales Territory].[Sales Territory Country].[Australia]

```

You can still run this query as a predefined MDX string. However, to build the same query using R using the `axis()` function, you must renumber the axes starting at 1.

## 2. Explore cubes and their fields on an SSAS instance

You can use the `explore` function to return a list of cubes, dimensions, or members to use in constructing your query. This is handy if you don't have access to other OLAP browsing tools, or if you want to programmatically manipulate or construct the MDX query.

#### To list the cubes available on the specified connection

To view all cubes or perspectives on the instance that you have permission to view, provide the handle as an argument to `explore`.

#### IMPORTANT

The final result is **not** a cube; TRUE merely indicates that the metadata operation was successful. An error is thrown if arguments are invalid.

```

cnnstr <- "Data Source=localhost; Provider=MSOLAP;"

ocs <- OlapConnection(cnnstr)

explore(ocs)

```

## RESULTS

*Analysis Services Tutorial*

*Internet Sales*

*Reseller Sales*

*Sales Summary*

[1] TRUE

### To get a list of cube dimensions

To view all dimensions in the cube or perspective, specify the cube or perspective name.

```
cnnstr <- "Data Source=localhost; Provider=MSOLAP;"
ocs \<- OlapConnection(cnnstr)
explore(ocs, "Sales")
```

## RESULTS

*Customer*

*Date*

*Region*

### To return all members of the specified dimension and hierarchy

After defining the source and creating the handle, specify the cube, dimension, and hierarchy to return. In the return results, items that are prefixed with -> represent children of the previous member.

```
cnnstr <- "Data Source=localhost; Provider=MSOLAP;"
ocs \<- OlapConnection(cnnstr)
explore(ocs, "Analysis Services Tutorial", "Product", "Product Categories", "Category")
```

## RESULTS

*Accessories*

*Bikes*

*Clothing*

*Components*

-> Assembly Components

-> Assembly Components

## See also

[Using data from OLAP cubes in R](#)



# Introducing revoscalepy in SQL Server Machine Learning

4/11/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

**revoscalepy** is a new Python library provided by Microsoft to support distributed computing, remote compute contexts, and high-performance algorithms for Python developers.

It is based on the **RevoScaleR** package for R, which was provided in Microsoft R Server and SQL Server R Services, and aims to provide the same functionality:

- Supports multiple compute contexts, both remote and local
- Provides functions equivalent to those in RevoScaleR for data transformation and visualization
- Provides Python versions of RevoScaleR machine learning algorithms for distributed or parallel processing
- Improved performance, including use of the Intel math libraries

MicrosoftML packages are also provided for both R and Python. For more information, see [Using MicrosoftML in SQL Server](#)

## Versions and supported platforms

The **revoscalepy** module is available only when you install one of the following Microsoft products:

- Machine Learning Services, in SQL Server 2017
- Microsoft Machine Learning Server 9.2.0 or later

To get the latest version of revoscalepy, install Cumulative Update 1 for SQL Server 2017. It includes many improvements in Python, including:

- A new Python function, `rx_create_col_info`, that gets schema information from a SQL Server data source, like `rxCreateCollInfo` for
- Enhancements to `rx_exec` to support parallel scenarios using the `RxLocalParallel` compute context.

## Supported functions and data types

This section provides an overview of the Python data types and new Python functions supported in the **revoscalepy** module, beginning with SQL Server 2017 CTP 2.0 release.

For the latest list of functions in the Python libraries released to date, see these links:

- [revoscalepy for Python](#)
- [microsoftml for Python](#)

### Data types, data sources, and compute contexts

SQL Server and Python use different data types in some cases. For a list of mappings between SQL and Python data types, see [Python libraries and data Types](#).

Data sources supported for machine learning with Python in SQL Server includes ODBC data sources, SQL Server database, and local files, including XDF files.

You create the data source object by using functions listed in the following table. After defining the data source, you load or transform the data by using an appropriate [ETL function](#).

#### IMPORTANT

Many function names have changed since the initial release of Python in CTP 2.0.

### SQL Server data

- Use `RxSqlServerData` to define a data source from a query or table
- Use `RxInSqlServer` to create a SQL Server compute context
- Use `RxOdbcData` to create a data source from an ODBC connection

**revoscalepy** also supports the [XDF data source](#), used for moving data between memory and other data sources.

#### TIP

If you are new to the idea of data sources or compute contexts, we recommend that you start by reading about distributed computing works for machine learning in [RevoScaleR](#).

### Machine learning and summary functions

The following machine learning algorithms and summary functions from RevoScaleR are included in SQL Server 2017, beginning with CTP 2.0.

FUNCTION	DESCRIPTION	NOTES
<code>rx_btrees</code>	Fit stochastic gradient boosted decision trees	<code>rx_btrees_ex</code> in CTP 2.0
<code>rx_dforest</code>	Fit classification and regression decision forests	<code>rx_dforest_ex</code> in CTP 2.0
<code>rx_dtree</code>	Fit classification and regression trees	<code>rx_dtree_ex</code> in CTP 2.0
<code>rx_lin_mod</code>	Create a linear model	<code>rx_lin_mod_ex</code> in CTP 2.0
<code>rx_logit</code>	Create a logistic regression model	<code>rx_logit_ex</code> in CTP 2.0
<code>rx_predict</code>	Generate predictions from a trained model	<code>rx_predict_ex</code> in CTP 2.0
<code>rx_summary</code>	Generate a summary of the model	

New machine learning algorithms are also provided by the Python version of [MicrosoftML](#):

FUNCTION	DESCRIPTION
<code>rx_fast_forest</code>	Create a decision forest model
<code>rx_fast_linear</code>	Linear regression with stochastic dual coordinate ascent

FUNCTION	DESCRIPTION
<code>rx_fast_trees</code>	Create a boosted tree model
<code>rx_logistic_regression</code>	Create a logistic regression model
<code>rx_neural_network</code>	Create a customizable neural network model
<code>rx_onesclass_svm</code>	Creates a SVM model on an imbalanced dataset, for use in anomaly detection

#### TIP

Many of these algorithms are already provided as modules in Azure Machine Learning.

MicrosoftML for Python also includes a variety of transformations and helper functions, such as:

- `rx_predict` generates predictions from a trained model and can be used for realtime scoring
- image featurization functions
- functions for text processing and sentiment extraction

For details, see [Introduction to MicrosoftML](#)

#### NOTE

The Python community uses coding conventions that might be different than what you're used to, including all lowercase letters and underscores rather than camel casing for parameter names. Also, maybe you've noticed that the **revoscalepy** library is always lowercase. That's right! Another Python convention.

Check out the tips on the Python reference documentation for Microsoft R: [Python function help][Python function help](#)

## Examples

You can run code that includes **revoscalepy** functions either locally or in a remote compute context. You can also run Python inside SQL Server by embedding Python code in a stored procedure.

When running locally, you typically run a Python script from the command line, or from a Python development environment, and specify a SQL Server compute context using one of the **revoscalepy** functions. You can use the remote compute context for the entire code, or for individual functions. For example, you might want to offload model training to the server to use the latest data and avoid data movement.

If you want to put a complete Python script inside the stored procedure, `sp_execute_external_script`, we recommend that you rewrite the code as a single function that has clearly defined inputs and outputs. Inputs and outputs must be **pandas** data frames. When this is done, you can call the stored procedure from any client that supports T-SQL, easily pass SQL queries as inputs, and save the results to SQL tables. For an example, see [In-Database Python Analytics for SQL Developers](#).

### Using remote compute contexts

- [Create a Model using revoscalepy](#)

This example demonstrates how to run Python using an instance of SQL Server as the compute context.

### Using a stored procedure

- [Run Python using T-SQL](#)

This example demonstrates the basic mechanism of calling Python script that is embedded in a stored procedure.

## Using revoscalepy with MicrosoftML

The Python functions for MicrosoftML are integrated with the compute contexts and data sources that are provided in revoscalepy. Therefore, you could use an MicrosoftML algorithm to define and train a model in Python, and use the revoscalepy functions to execute the Python code either locally or in a SQL Server compute context.

Just import the modules in your Python code, and then reference the individual functions you need.

```
from microsoftml.modules.logistic_regression.rx_logistic_regression import rx_logistic_regression
from revoscalepy.functions.RxSummary import rx_summary
from revoscalepy.etl.RxImport import rx_import_datasource
```

## Requirements

To run Python code in SQL Server, you must have installed SQL Server 2017 together with the feature **Machine Learning Services**, and enabled the Python language. Earlier versions of SQL Server do not support Python integration.

### NOTE

Open source distributions of Python do not support SQL Server compute contexts. However, if you need to publish and consume Python applications from Windows, you can install Microsoft Machine Learning Server without installing SQL Server. For more information, see [Install SQL Server 2017 Machine Learning Server \(Standalone\)](#).

## Get more help

Complete documentation for these APIs will be available when the product is released. In the meantime, we recommend that you reference the corresponding function in the RevoScaleR or MicrosoftML libraries.

- [RevoScaleR](#).
- [MicrosoftML](#)

You can get help on any Python function by importing the module, and then calling `help()`. For example, running `help(revoscalepy)` from your Python IDE returns a list of all functions in the revoscalepy module, with their signatures.

If you use Python Tools for Visual Studio, you can use IntelliSense to get syntax and argument help. For more information, see [Python support in Visual Studio](#), and download the extension that matches your version of Visual Studio. You can use Python with Visual Studio 2015 and 2017, or earlier versions.

## See Also

[Python tutorials](#)

# Known issues in Machine Learning Services

6/20/2018 • 24 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes known problems or limitations with machine learning components that are provided as an option in SQL Server 2016 and SQL Server 2017.

The information here applies to all of the following, unless otherwise indicated:

SQL Server 2016

- R Services (In-Database)
- Microsoft R Server (Standalone)

SQL Server 2017

- Machine Learning Services for R (In-Database)
- Machine Learning Services for Python (In-Database)
- Machine Learning Server (Standalone)

## Setup and configuration issues

For a description of processes and common questions that are related to initial setup and configuration, see [Upgrade and installation FAQ](#). It contains information about upgrades, side-by-side installation, and installation of new R or Python components.

### R Script runtime error (SQL Server 2017 CU5-CU7 Regression)

For SQL Server 2017, in cumulative updates 5 through 7, there is a regression in the **rlauncher.config** file where the temp directory file path includes a space. This regression is corrected in CU8.

The error you will see when running R script includes the following messages:

*Unable to communicate with the runtime for 'R' script. Please check the requirements of 'R' runtime.*

STDERR message(s) from external script:

*Fatal error: cannot create 'R\_TempDir'*

### Workaround

Apply CU8 when it becomes available. Alternatively, you can recreate **rlauncher.config** by running **registerrext** with uninstall/install on an elevated command prompt.

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe /uninstall /sqlbinnpath:
<SQLInstanceBinnPath> /userpoolsize:0 /instance:<SQLInstanceName>

<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe /install /sqlbinnpath:
<SQLInstanceBinnPath> /userpoolsize:0 /instance:<SQLInstanceName>
```

The following example shows the commands with the default instance "MSSQL14.MSSQLSERVER" installed into "C:\Program Files\Microsoft SQL Server":

```
"C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRext.exe" /uninstall
/sqlbinnpath:"C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Binn" /userpoolsize:0
/instance:MSSQLSERVER

"C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRext.exe" /install
/sqlbinnpath:"C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Binn" /userpoolsize:0
/instance:MSSQLSERVER
```

## Unable to install SQL Server machine learning features on a domain controller

If you try to install SQL Server 2016 R Services or SQL Server 2017 Machine Learning Services on a domain controller, setup fails, with these errors:

*An error occurred during the setup process of the feature*

*Cannot find group with identity*

*Component error code: 0x80131509*

The failure occurs because, on a domain controller, the service cannot create the 20 local accounts required to run machine learning. In general, we do not recommend installing SQL Server on a domain controller. For more information, see [Support bulletin 2032911](#).

## Install the latest service release to ensure compatibility with Microsoft R Client

If you install the latest version of Microsoft R Client and use it to run R on SQL Server in a remote compute context, you might get an error like the following:

*You are running version 9.x.x of Microsoft R Client on your computer, which is incompatible with Microsoft R Server version 8.x.x. Download and install a compatible version.*

SQL Server 2016 requires that the R libraries on the client exactly match the R libraries on the server. The restriction has been removed for releases later than R Server 9.0.1. However, if you encounter this error, verify the version of the R libraries that's used by your client and the server and, if necessary, update the client to match the server version.

The version of R that is installed with SQL Server R Services is updated whenever a SQL Server service release is installed. To ensure that you always have the most up-to-date versions of R components, be sure to install all service packs.

To ensure compatibility with Microsoft R Client 9.0.0, install the updates that are described in this [support article](#).

To avoid problems with R packages, you can also upgrade the version of the R libraries that are installed on the server, by changing your servicing agreement to use the Modern Lifecycle Support policy, as described in [the next section](#). When you do so, the version of R that's installed with SQL Server is updated on the same schedule used for updates of machine Learning Server (formerly Microsoft R Server).

**Applies to:** SQL Server 2016 R Services, with R Server version 9.0.0 or earlier

## R components missing from CU3 setup

A limited number of Azure virtual machines were provisioned without the R installation files that should be included with SQL Server. The issue applies to virtual machines provisioned in the period from 2018-01-05 to 2018-01-23. This issue might also affect on-premises installations, if you applied the CU3 update for SQL Server 2017 during the period from 2018-01-05 to 2018-01-23.

A service release has been provided that includes the correct version of the R installation files.

- [Cumulative Update Package 3 for SQL Server 2017 KB4052987](#).

To install the components and repair SQL Server 2017 CU3, you must uninstall CU3, and reinstall the updated version:

1. Download the updated CU3 installation file, which includes the R installers.
2. Uninstall CU3. In Control Panel, search for **Uninstall an update**, and then select "Hotfix 3015 for SQL Server 2017 (KB4052987) (64-bit)". Proceed with uninstall steps.
3. Reinstall the CU3 update, by double-clicking on the update for KB4052987 that you just downloaded:  
`SQLServer2017-KB4052987-x64.exe`. Follow the installation instructions.

#### **Unable to install Python components in offline installations of SQL Server 2017 CTP 2.0 or later**

If you install a pre-release version of SQL Server 2017 on a computer without internet access, the installer might fail to display the page that prompts for the location of the downloaded Python components. In such an instance, you can install the Machine Learning Services feature, but not the Python components.

This issue is fixed in the release version. Also, this limitation does not apply to R components.

**Applies to:** SQL Server 2017 with Python

#### **Warning of incompatible version when you connect to an older version of SQL Server R Services from a client by using SQL Server 2017 (14.x)**

When you run R code in a SQL Server 2016 compute context, you might see the following error:

*You are running version 9.0.0 of Microsoft R Client on your computer, which is incompatible with the Microsoft R Server version 8.0.3. Download and install a compatible version.*

This message is displayed if either of the following two statements is true,

- You installed R Server (Standalone) on a client computer by using the setup wizard for SQL Server 2017 (14.x).
- You installed Microsoft R Server by using the [separate Windows installer](#).

To ensure that the server and client use the same version you might need to use *binding*, supported for Microsoft R Server 9.0 and later releases, to upgrade the R components in SQL Server 2016 instances. To determine if support for upgrades is available for your version of R Services, see [Upgrade an instance of R Services using SqlBindR.exe](#).

**Applies to:** SQL Server 2016 R Services, with R Server version 9.0.0 or earlier

#### **Setup for SQL Server 2016 service releases might fail to install newer versions of R components**

When you install a cumulative update or install a service pack for SQL Server 2016 on a computer that is not connected to the internet, the setup wizard might fail to display the prompt that lets you update the R components by using downloaded CAB files. This failure typically occurs when multiple components were installed together with the database engine.

As a workaround, you can install the service release by using the command line and specifying the

`MRCACHEDIRECTORY` argument as shown in this example, which installs CU1 updates:

```
C:\<path to installation media>\SQLServer2016-KB3164674-x64.exe /Action=Patch /IACCEPTOPENLICENSETERMS
/MRCACHEDIRECTORY=<path to CU1 CAB files>
```

To get the latest installers, see [Install machine learning components without internet access](#).

**Applies to:** SQL Server 2016 R Services, with R Server version 9.0.0 or earlier

#### **Launchpad services fails to start if the version is different from the R version**

If you install SQL Server R Services separately from the database engine, and the build versions are different, you might see the following error in the System Event log:

*The SQL Server Launchpad service failed to start due to the following error: The service did not respond to the start or control request in a timely fashion.*

For example, this error might occur if you install the database engine by using the release version, apply a patch to upgrade the database engine, and then add the R Services feature by using the release version.

To avoid this problem, use a utility such as File Manager to compare the versions of Launchpad.exe with version of SQL binaries, such as sqldk.dll. All components should have the same version number. If you upgrade one component, be sure to apply the same upgrade to all other installed components.

Look for Launchpad in the `Binn` folder for the instance. For example, in a default installation of SQL Server 2016, the path might be `C:\Program Files\Microsoft SQL Server\MSSQL.13.InstanceNameMSSQL\Binn`.

### **Remote compute contexts are blocked by a firewall in SQL Server instances that are running on Azure virtual machines**

If you have installed SQL Server 2017 on a Windows Azure virtual machine, you might not be able to use compute contexts that require the use of the virtual machine's workspace. The reason is that, by default, the firewall on Azure virtual machines includes a rule that blocks network access for local R user accounts.

As a workaround, on the Azure VM, open **Windows Firewall with Advanced Security**, select **Outbound Rules**, and disable the following rule: **Block network access for R local user accounts in SQL Server instance MSSQLSERVER**. You can also leave the rule enabled, but change the security property to **Allow if secure**.

### **Implied authentication in SQLEXPRESS**

When you run R jobs from a remote data-science workstation by using Integrated Windows authentication, SQL Server uses *implied authentication* to generate any local ODBC calls that might be required by the script. However, this feature did not work in the RTM build of SQL Server Express Edition.

To fix the issue, we recommend that you upgrade to a later service release.

If upgrade is not feasible, as a workaround, use a SQL login to run remote R jobs that might require embedded ODBC calls.

**Applies to:** SQL Server 2016 R Services Express Edition

### **Performance limits when libraries used by SQL Server are called from other tools**

It is possible to call the machine learning libraries that are installed for SQL Server from an external application, such as RGui. Doing so might be the most convenient way to accomplish certain tasks, such as installing new packages, or running ad hoc tests on very short code samples. However, outside of SQL Server, performance might be limited.

For example, even if you are using the Enterprise Edition of SQL Server, R runs in single-threaded mode when you run your R code by using external tools. To get the benefits of performance in SQL Server, initiate a SQL Server connection and use `sp_execute_external_script` to call the external script runtime.

In general, avoid calling the machine learning libraries that are used by SQL Server from external tools. If you need to debug R or Python code, it is typically easier to do so outside of SQL Server. To get the same libraries that are in SQL Server, you can install Microsoft R Client, [SQL Server 2017 Machine Learning Server \(Standalone\)](#), or [SQL Server 2016 R Server \(Standalone\)](#).

### **SQL Server Data Tools does not support permissions required by external scripts**

When you use Visual Studio or SQL Server Data Tools to publish a database project, if any principal has permissions specific to external script execution, you might get an error like this one:

*TSQL Model: Error detected when reverse engineering the database. The permission was not recognized and was not imported.*

Currently the DACPAC model does not support the permissions used by R Services or Machine Learning Services, such as GRANT ANY EXTERNAL SCRIPT, or EXECUTE ANY EXTERNAL SCRIPT. This issue will be fixed in a later release.

As a workaround, run the additional GRANT statements in a post-deployment script.

#### **External script execution is throttled due to resource governance default values**

In Enterprise Edition, you can use resource pools to manage external script processes. In some early release builds, the maximum memory that could be allocated to the R processes was 20 percent. Therefore, if the server had 32 GB of RAM, the R executables (RTerm.exe and BxlServer.exe) could use a maximum of 6.4 GB in a single request.

If you encounter resource limitations, check the current default. If 20 percent is not enough, see the documentation for SQL Server on how to change this value.

**Applies to:** SQL Server 2016 R Services, Enterprise Edition

## R script execution issues

This section contains known issues that are specific to running R on SQL Server, as well as some issues that are related to the R libraries and tools published by Microsoft, including RevoScaleR.

For additional known issues that might affect R solutions, see the [Machine Learning Server](#) site.

#### **Access denied warning when executing R scripts on SQL Server in a non default location**

If the instance of SQL Server has been installed to a non-default location, such as outside the `Program Files` folder, the warning ACCESS\_DENIED is raised when you try to run scripts that install a package. For example:

```
In normalizePath(path.expand(path), winslash, mustWork) : path[2] = "ExternalLibraries/R/8/1": Access is denied
```

The reason is that an R function attempts to read the path, and fails if the built-in users group **SQLRUUserGroup**, does not have read access. The warning that is raised does not block execution of the current R script, but the warning might recur repeatedly whenever the user runs any other R script.

If you have installed SQL Server to the default location, this error does not occur, because all Windows users have read permissions on the `Program Files` folder.

This issue is addressed in an upcoming service release. As a workaround, provide the group, **SQLRUUserGroup**, with read access for all parent folders of `ExternalLibraries`.

#### **Serialization error between old and new versions of RevoScaleR**

When you pass a model using a serialized format to a remote SQL Server instance, you might get the error:

```
Error in memDecompress(data, type = decompress) internal error -3 in memDecompress(2).
```

This error is raised if you saved the model using a recent version of the serialization function, `rxSerializeModel`, but the SQL Server instance where you deserialize the model has an older version of the RevoScaleR APIs, from SQL Server 2017 CU2 or earlier.

As a workaround, you can upgrade the SQL Server 2017 instance to CU3 or later.

The error does not appear if the API version is the same, or if you are moving a model saved with an older serialization function to a server that uses a newer version of the serialization API.

In other words, use the same version of RevoScaleR for both serialization and deserialization operations.

#### **Real-time scoring does not correctly handle the `learningRate` parameter in tree and forest models**

If you create a model using a decision tree or decision forest method and specify the learning rate, you might see inconsistent results when using `sp_rxpredict` or the SQL `PREDICT` function, as compared to using `rxPredict`.

The cause is an error in the API that processes serialized models, and is limited to the `learningRate` parameter: for example, in [rxBTrees](#), or

This issue is addressed in an upcoming service release.

### Limitations on processor affinity for R jobs

In the initial release build of SQL Server 2016, you could set processor affinity only for CPUs in the first k-group. For example, if the server is a 2-socket machine with two k-groups, only processors from the first k-group are used for the R processes. The same limitation applies when you configure resource governance for R script jobs.

This issue is fixed in SQL Server 2016 Service Pack 1. We recommend that you upgrade to the latest service release.

**Applies to:** SQL Server 2016 R Services RTM version

### Changes to column types cannot be performed when reading data in a SQL Server compute context

If your compute context is set to the SQL Server instance, you cannot use the `colClasses` argument (or other similar arguments) to change the data type of columns in your R code.

For example, the following statement would result in an error if the column `CRSDepTimeStr` is not already an integer:

```
data <- RxSqlServerData(
 sqlQuery = "SELECT CRSDepTimeStr, ArrDelay FROM AirlineDemoSmall",
 connectionString = connectionString,
 colClasses = c(CRSDepTimeStr = "integer"))
```

As a workaround, you can rewrite the SQL query to use `CAST` or `CONVERT` and present the data to R by using the correct data type. In general, performance is better when you work with data by using SQL rather than by changing data in the R code.

**Applies to:** SQL Server 2016 R Services

### Limits on size of serialized models

When you save a model to a SQL Server table, you must serialize the model and save it in a binary format. Theoretically the maximum size of a model that can be stored with this method is 2 GB, which is the maximum size of varbinary columns in SQL Server.

If you need to use larger models, the following workarounds are available:

- Take steps to reduce the size of your model. Some open source R packages include a great deal of information in the model object, and much of this information can be removed for deployment.
- Use feature selection to remove unnecessary columns.
- If you are using an open source algorithm, consider a similar implementation using the corresponding algorithm in MicrosoftML or RevoScaleR. These packages have been optimized for deployment scenarios.
- After the model has been rationalized and the size reduced using the preceding steps, see if the `memCompress` function in base R can be used to reduce the size of the model before passing it to SQL Server. This option is best when the model is close to the 2 GB limit.
- For larger models, you can use the SQL Server [FileTable](#) feature to store the models, rather than using a varbinary column.

To use FileTables, you must add a firewall exception, because data stored in FileTables is managed by the Filestream filesystem driver in SQL Server, and default firewall rules block network file access. For more

information, see [Enable Prerequisites for FileTable](#).

After you have enabled FileTable, to write the model, you get a path from SQL using the FileTable API, and then write the model to that location from your code. When you need to read the model, you get the path from SQL and then call the model using the path from your script. For more information, see [Access FileTables with File Input-Output APIs](#).

### Avoid clearing workspaces when you execute R code in a SQL Server compute context

If you use an R command to clear your workspace of objects while running R code in a SQL Server compute context, or if you clear the workspace as part of an R script called by using `sp_execute_external_script`, you might get this error: *workspace object revoScriptConnection not found*

`revoScriptConnection` is an object in the R workspace that contains information about an R session that is called from SQL Server. However, if your R code includes a command to clear the workspace (such as `rm(list=ls())`), all information about the session and other objects in the R workspace is cleared as well.

As a workaround, avoid indiscriminate clearing of variables and other objects while you're running R in SQL Server. Although clearing the workspace is common when working in the R console, it can have unintended consequences.

- To delete specific variables, use the R `remove` function: for example, `remove('name1', 'name2', ...)`
- If there are multiple variables to delete, save the names of temporary variables to a list and perform periodic garbage collection.

### Restrictions on data that can be provided as input to an R script

You cannot use in an R script the following types of query results:

- Data from a Transact-SQL query that references AlwaysEncrypted columns.
- Data from a Transact-SQL query that references masked columns.

If you need to use masked data in an R script, a possible workaround is to make a copy of the data in a temporary table and use that data instead.

### Use of strings as factors can lead to performance degradation

Using string type variables as factors can greatly increase the amount of memory used for R operations. This is a known issue with R in general, and there are many articles on the subject. For example, see [Factors are not first-class citizens in R, by John Mount, in R-bloggers](#) or [stringsAsFactors: An unauthorized biography](#), by Roger Peng.

Although the issue is not specific to SQL Server, it can greatly affect performance of R code run in SQL Server. Strings are typically stored as varchar or nvarchar, and if a column of string data has many unique values, the process of internally converting these to integers and back to strings by R can even lead to memory allocation errors.

If you do not absolutely require a string data type for other operations, mapping the string values to a numeric (integer) data type as part of data preparation would be beneficial from a performance and scale perspective.

For a discussion of this issue, and other tips, see [Performance for R Services - data optimization](#).

### Arguments `varsToKeep` and `varsToDelete` are not supported for SQL Server data sources

When you use the `rxDatStep` function to write results to a table, using the `varsToKeep` and `varsToDelete` is a handy way of specifying the columns to include or exclude as part of the operation. However, these arguments are not supported for SQL Server data sources.

### Limited support for SQL data types in `sp_execute_external_script`

Not all data types that are supported in SQL can be used in R. As a workaround, consider casting the unsupported data type to a supported data type before passing the data to `sp_execute_external_script`.

For more information, see [R libraries and data types](#).

## Possible string corruption

Any round trip of string data from Transact-SQL to R and then to Transact-SQL again can result in corruption. This is due to the different encodings used in R and in SQL Server, as well as the various collations and languages that are supported in R and Transact-SQL. Any string in a non-ASCII encoding can potentially be handled incorrectly.

When you send string data to R, convert it to an ASCII representation, if possible.

This limitation applies to data passed between SQL Server and Python as well. Multibyte characters should be passed as UTF-8 and stored as Unicode.

## Only one value of type `raw` can be returned from `sp_execute_external_script`

When a binary data type (the R `raw` data type) is returned from R, the value must be sent in the output data frame.

With data types other than `raw`, you can return parameter values along with the results of the stored procedure by adding the `OUTPUT` keyword. For more information, see [Parameters](#).

If you want to use multiple output sets that include values of type `raw`, one possible workaround is to do multiple calls of the stored procedure, or to send the result sets back to SQL Server by using ODBC.

## Loss of precision

Because Transact-SQL and R support various data types, numeric data types can suffer loss of precision during conversion.

For more information about implicit data-type conversion, see [R libraries and data types](#).

## Variable scoping error when you use the `transformFunc` parameter

To transform data while you are modeling, you can pass a `transformFunc` argument in a function such as `rxLinmod` or `rxLogit`. However, nested function calls can lead to scoping errors in the SQL Server compute context, even if the calls work correctly in the local compute context.

*The sample data set for the analysis has no variables*

For example, assume that you have defined two functions, `f` and `g`, in your local global environment, and `g` calls `f`. In distributed or remote calls involving `g`, the call to `g` might fail with this error, because `f` cannot be found, even if you have passed both `f` and `g` to the remote call.

If you encounter this problem, you can work around the issue by embedding the definition of `f` inside your definition of `g`, anywhere before `g` would ordinarily call `f`.

For example:

```
f <- function(x) { 2*x * 3 }
g <- function(y) {
 a <- 10 * y
 f(a)
}
```

To avoid the error, rewrite the definition as follows:

```
g <- function(y){
 f <- function(x) { 2*x +3}
 a <- 10 * y
 f(a)
}
```

## Data import and manipulation using RevoScaleR

When **varchar** columns are read from a database, white space is trimmed. To prevent this, enclose strings in non-white-space characters.

When functions such as `rxDataStep` are used to create database tables that have **varchar** columns, the column width is estimated based on a sample of the data. If the width can vary, it might be necessary to pad all strings to a common length.

Using a transform to change a variable's data type is not supported when repeated calls to `rxImport` or `rxTextToXdf` are used to import and append rows, combining multiple input files into a single .xdf file.

### Limited support for rxExec

In SQL Server 2016, the `rxExec` function that's provided by the RevoScaleR package can be used only in single-threaded mode.

Parallelism for `rxExec` across multiple processes is planned for a future release.

### Increase the maximum parameter size to support rxGetVarInfo

If you use data sets with extremely large numbers of variables (for example, over 40,000), set the `max-ppsize` flag when you start R to use functions such as `rxGetVarInfo`. The `max-ppsize` flag specifies the maximum size of the pointer protection stack.

If you are using the R console (for example, RGui.exe or RTerm.exe), you can set the value of *max-ppsize* to 500000 by typing:

```
R --max-ppsize=500000
```

## Issues with the rxDTTree function

The `rxDTTree` function does not currently support in-formula transformations. In particular, using the `F()` syntax for creating factors on the fly is not supported. However, numeric data is automatically binned.

Ordered factors are treated the same as factors in all RevoScaleR analysis functions except `rxDTTree`.

## Python script execution issues

This section contains known issues that are specific to running Python on SQL Server, as well as issues that are related to the Python packages published by Microsoft, including [revoscalepy](#) and [microsoftml](#).

### Call to pretrained model fails if path to model is too long

If you installed the pretrained models in an early release of SQL Server 2017, the complete path to the trained model file might be too long for Python to read. This limitation is fixed in a later service release.

There are several potential workarounds:

- When you install the pretrained models, choose a custom location.
- If possible, install the SQL Server instance under a custom installation path with a shorter path, such as C:\SQL\MSSQL14.MSSQLSERVER.
- Use the Windows utility [Fsutil](#) to create a hard link that maps the model file to a shorter path.
- Update to the latest service release.

### Error when saving serialized model to SQL Server

When you pass a model to a remote SQL Server instance, and try to read the binary model using the `rx_serialize` function in [revoscalepy](#), you might get the error:

```
NameError: name 'rx_serialize_model' is not defined
```

This error is raised if you saved the model using a recent version of the serialization function, but the SQL Server instance where you deserialize the model does not recognize the serialization API.

To resolve the issue, upgrade the SQL Server 2017 instance to CU3 or later.

#### Failure to initialize a varbinary variable causes an error in BxlServer

If you run Python code in SQL Server using `sp_execute_external_script`, and the code has output variables of type varbinary(max), varchar(max) or similar types, the variable must be initialized or set as part of your script. Otherwise, the data exchange component, BxlServer, raises an error and stops working.

This limitation will be fixed in an upcoming service release. As a workaround, make sure that the variable is initialized within the Python script. Any valid value can be used, as in the following examples:

```
declare @b varbinary(max);
exec sp_execute_external_script
 @language = N'Python'
 , @script = N'b = 0x0'
 , @params = N'@b varbinary(max) OUTPUT'
 , @b = @b OUTPUT;
go
```

```
declare @b varchar(30);
exec sp_execute_external_script
 @language = N'Python'
 , @script = N' b = "" '
 , @params = N'@b varchar(30) OUTPUT'
 , @b = @b OUTPUT;
go
```

#### Telemetry warning on successful execution of Python code

Beginning with SQL Server 2017 CU2, the following message might appear even if Python code otherwise runs successfully:

```
STDERR message(s) from external script: *~PYTHON_SERVICES\lib\site-
packages\revoscalepy\utils\RxTelemetryLogger SyntaxWarning: telemetry_state is used prior to global
declaration
```

This issue has been fixed in SQL Server 2017 Cumulative Update 3 (CU3).

## Revolution R Enterprise and Microsoft R Open

This section lists issues specific to R connectivity, development, and performance tools that are provided by Revolution Analytics. These tools were provided in earlier pre-release versions of SQL Server 2017.

In general, we recommend that you uninstall these previous versions and install the latest version of SQL Server or Microsoft R Server.

#### Revolution R Enterprise is not supported

Installing Revolution R Enterprise side by side with any version of R Services (In-Database) is not supported.

If you have an existing license for Revolution R Enterprise, you must put it on a separate computer from both the SQL Server instance and any workstation that you want to use to connect to the SQL Server instance.

Some pre-release versions of R Services (In-Database) included an R development environment for Windows that was created by Revolution Analytics. This tool is no longer provided, and is not supported.

For compatibility with R Services (In-Database), we recommend that you install Microsoft R Client instead. [R Tools for Visual Studio](#) and [Visual Studio Code](#) also supports Microsoft R solutions.

### **Compatibility issues with SQLite ODBC driver and RevoScaleR**

Revision 0.92 of the SQLite ODBC driver is incompatible with RevoScaleR. Revisions 0.88-0.91 and 0.93 and later are known to be compatible.

## See also

[What's new in SQL Server 2016](#)

[Troubleshooting machine learning in SQL Server](#)

# Install SQL Server machine learning features on an Azure virtual machine

5/29/2018 • 3 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

We recommend using the [Data Science virtual machine](#), but if you want a VM that has just SQL Server 2017 Machine Learning Services or SQL Server 2016 R Services, this article guides you through the steps.

## Create a virtual machine on Azure

1. In the Azure portal in the left-side list, click **Virtual machines** and then click **Add**.
2. Search for SQL Server 2017 Enterprise Edition or SQL Server 2016 Enterprise Edition.
3. Configure the server name and account permissions, and select a pricing plan.
4. In **SQL Server Settings** (Step 4 in the VM setup wizard), locate **Machine Learning Services (Advanced Analytics)** (or **R Services** for SQL Server 2016) and click **Enable**.
5. Review the summary presented for validation and click **OK**.
6. When the virtual machine is ready, connect to it, and open SQL Server Management Studio, which is pre-installed. Machine learning is ready to run.
7. To verify this, you can open a new query window and run a simple statement such as this one, which uses R to generate a sequence of numbers from 1 to 10.

```
EXEC sp_execute_external_script
@language = N'R'
, @script = N' OutputDataSet <- as.data.frame(seq(1, 10,));'
, @input_data_1 = N' ;'
WITH RESULT SETS (([Sequence] int NOT NULL));
```

8. If you plan to connect to the instance from a remote data science client, complete [additional steps](#) as necessary.

## Disable machine learning features on a SQL Server VM

You can also enable or disable the feature on an existing SQL Server virtual machine at any time.

1. Open the virtual machine blade.
2. Click **Settings**, and select **SQL Server configuration**.
3. Make changes to features and apply.

## Add R Services to an existing SQL Server 2016 Enterprise VM

If you created an Azure virtual machine that included SQL Server without machine learning, you can add the feature by following these steps:

1. Re-run SQL Server setup and add the feature on the **Server Configuration** page of the wizard.
2. Enable execution of external scripts and restart the SQL Server instance. For more information, see [Install SQL Server 2016 R Services](#).
3. (Optional) Configure database access for R worker accounts, if needed for remote script execution.
4. (Optional) Modify a firewall rule on the Azure virtual machine, if you intend to allow R script execution from remote data science clients. For more information, see [Unblock firewall](#).

5. Install or enable required network libraries. For more information, see [Add network protocols](#).

## Additional steps

Some additional steps are required if you expect remote clients to access the server as a remote SQL Server compute context.

### Unblock the firewall

By default, the firewall on the Azure virtual machine includes a rule that blocks network access for local user accounts.

You must disable this rule to ensure that you can access the SQL Server instance from a remote data science client. Otherwise, your machine learning code cannot execute in compute contexts that use the virtual machine's workspace.

To enable access from remote data science clients:

1. On the virtual machine, open Windows Firewall with Advanced Security.
2. Select **Outbound Rules**
3. Disable the following rule:

```
Block network access for R local user accounts in SQL Server instance MSSQLSERVER
```

### Enable ODBC callbacks for remote clients

If you expect that clients calling the server will need to issue ODBC queries as part of their machine learning solutions, you must ensure that the Launchpad can make ODBC calls on behalf of the remote client. To do this, you must allow the SQL worker accounts that are used by Launchpad to log into the instance. For more information, see [Install SQL Server 2016 R Services](#).

### Add network protocols

- Enable Named Pipes

R Services (In-Database) uses the Named Pipes protocol for connections between the client and server computers, and for some internal connections. If Named Pipes is not enabled, you must install and enable it on both the Azure virtual machine, and on any data science clients that connect to the server.

- Enable TCP/IP

TCP/IP is required for loopback connections. If you get the following error, enable TCP/IP on the virtual machine that supports the instance:

"DBNETLIB; SQL Server does not exist or access denied"

# Troubleshoot machine learning in SQL Server

6/1/2018 • 2 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Use this page as a starting point for working through known issues.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services (R and Python)

## Known issues

The following articles describe known issues with the current and previous releases:

- [Known issues for R Services](#)
- [SQL Server 2016 release notes](#)
- [SQL Server 2017 release notes](#)

## How to gather system information

If you have encountered an error, or need to understand an issue in your environment, it is important that you collect related information systematically. The following article provides a list of information that facilitates self-help troubleshooting, or a request for technical support.

- [Data collection for machine learning troubleshooting](#)

## Setup and configuration guides

Start here if you have not set up machine learning with SQL Server, or if you want to add the feature:

- [Install SQL Server 2017 Machine Learning Services \(In-Database\)](#)
- [Install SQL Server 2017 Machine Learning Server \(Standalone\)](#)
- [Install SQL Server 2016 R Services \(In-Database\)](#)
- [Install SQL Server 2016 R Server \(Standalone\)](#)
- [Command prompt setup](#)
- [Offline setup \(no internet\)](#)

## Configuration

The following articles contain information about defaults, and how to customize the configuration for machine learning on an instance:

- [Modify the user account pool for SQL Server R Services](#)
- [Configure and manage advanced analytics extensions](#)
- [How to create a resource pool](#)
- [Optimization for R workloads](#)

# Troubleshoot data collection for machine learning

6/1/2018 • 10 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article describes data collection methods you should use when attempting to resolve problems on your own or with the help of Microsoft customer support.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services (R and Python)

## SQL Server version and edition

SQL Server 2016 R Services is the first release of SQL Server to include integrated R support. SQL Server 2016 Service Pack 1 (SP1) includes several major improvements, including the ability to run external scripts. If you are a SQL Server 2016 customer, you should consider installing SP1 or later.

SQL Server 2017 added Python language integration. You cannot get Python feature integration in earlier releases.

For assistance getting edition and versions, see this article, which lists the build numbers for each of the [SQL Server versions](#).

Depending on the edition of SQL Server you're using, some machine learning functionality might be unavailable, or limited. The following articles list of machine learning features in Enterprise, Developer, Standard, and Express editions.

- [Editions and supported features of SQL Server](#)
- [R and Python features by editions of SQL Server](#)

## R language and tool versions

In general, the version of Microsoft R that is installed when you select the R Services feature or the Machine Learning Services feature is determined by the SQL Server build number. If you upgrade or patch SQL Server, you must also upgrade or patch its R components.

For a list of releases and links to R component downloads, see [Install machine learning components without internet access](#). On computers with internet access, the required version of R is identified and installed automatically.

It's possible to upgrade the R Server components separately from the SQL Server database engine, in a process known as binding. Therefore, the version of R that you use when you run R code in SQL Server might differ depending on both the installed version of SQL Server and whether you have migrated the server to the latest R version.

### Determine the R version

The easiest way to determine the R version is to get the runtime properties by running a statement such as the following:

```

exec sp_execute_external_script
 @language = N'R'
 , @script = N'
Transform R version properties to data.frame
OutputDataSet <- data.frame(
 property_name = c("R.version", "Revo.version"),
 property_value = c(R.Version()$version.string, Revo.version$version.string),
 stringsAsFactors = FALSE)
Retrieve properties like R.home, libPath & default packages
OutputDataSet <- rbind(OutputDataSet, data.frame(
 property_name = c("R.home", "libPaths", "defaultPackages"),
 property_value = c(R.home(), .libPaths(), paste(getOption("defaultPackages"), collapse=", ")),
 stringsAsFactors = FALSE)
)
'

WITH RESULT SETS ((PropertyName nvarchar(100), PropertyValue nvarchar(4000)));

```

#### TIP

If R Services is not working, try running only the R script portion from RGui.

As a last resort, you can open files on the server to determine the installed version. To do so, locate the rlauncher.config file to get the location of the R runtime and the current working directory. We recommend that you make and open a copy of the file so that you don't accidentally change any properties.

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.<instance_name>\MSSQL\Binn\rlauncher.config
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\MSSQL\Binn\rlauncher.config
```

To get the R version and RevoScaleR versions, open an R command prompt, or open the RGui that's associated with the instance.

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.<instancename>\R_SERVICES\bin\x64\RGui.exe
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\R_SERVICES\bin\x64\RGui.exe
```

The R console displays the version information on startup. For example, the following version represents the default configuration for SQL Server 2017 CTP 2.0:

```

Microsoft R Open 3.3.3
The enhanced R distribution from Microsoft
Microsoft packages Copyright (C) 2017 Microsoft
Loading Microsoft R Server packages, version 9.1.0.
```

## Python versions

There are several ways to get the Python version. The easiest way is to run this statement from Management Studio or any other SQL query tool:

```
-- Get Python runtime properties:
exec sp_execute_external_script
 @language = N'Python'
 , @script = N'
import sys
import pkg_resources
OutputDataSet = pandas.DataFrame(
 {"property_name": ["Python.home", "Python.version", "Revo.version", "libpaths"],
 "property_value": [sys.executable[-10:], sys.version,
pkg_resources.get_distribution("revoscalepy").version, str(sys.path)]}
)
'

with RESULT SETS (SQL keywords) ((PropertyName nvarchar(100), PropertyValue nvarchar(4000)));

```

If Machine Learning Services is not running, you can determine the installed Python version by looking at the `pythonlauncher.config` file. We recommend that you make and open a copy of the file so that you don't accidentally change any properties.

1. For SQL Server 2017 only:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.
<instance_name>\MSSQL\Log\ExtensibilityLog\pythonlauncher.config
```

2. Get the value for **PYTHONHOME**.

3. Get the value of the current working directory.

#### **NOTE**

If you have installed both Python and R in SQL Server 2017, the working directory and the pool of worker accounts are shared for the R and Python languages.

## Are multiple instances of R or Python installed?

Check to see whether more than one copy of the R libraries is installed on the computer. This duplication can happen if:

- During setup you select both R Services (In-Database) and R Server (Standalone).
- You install Microsoft R Client in addition to SQL Server.
- A different set of R libraries was installed by using R Tools for Visual Studio, R Studio, Microsoft R Client, or another R IDE.
- The computer hosts multiple instances of SQL Server, and more than one instance uses machine learning.

The same conditions apply to Python.

If you find that multiple libraries or runtimes are installed, make sure that you get only the errors associated with the Python or R runtimes that are used by the SQL Server instance.

## Origin of errors

The errors that you see when you attempt to run R code can come from any of the following sources:

- SQL Server database engine, including the stored procedure `sp_execute_external_script`
- The SQL Server Trusted Launchpad
- Other components of the extensibility framework, including R and Python launchers and satellite processes
- Providers, such as Microsoft Open Database Connectivity (ODBC)
- R language

When you work with the service for the first time, it can be difficult to tell which messages originate from which

services. We recommend that you capture not only the exact message text, but the context in which you saw the message. Note the client software that you're using to run machine learning code:

- Are you using Management Studio? An external application?
- Are you running R code in a remote client, or directly in a stored procedure?

## SQL Server log files

Get the most recent SQL Server ERRORLOG. The complete set of error logs consists of the files from the following default log directory:

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.SQL2016\MSSQL\Log\ExtensibilityLog
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.SQL2016\MSSQL\Log\ExtensibilityLog
```

### NOTE

The exact folder name differs depending on the instance name.

## Errors returned by sp\_execute\_external\_script

Get the complete text of errors that are returned, if any, when you run the sp\_execute\_external\_script command.

To remove R or Python problems from consideration, you can run this script, which starts the R or Python runtime and passes data back and forth.

### For R

```
exec sp_execute_external_script @language =N'R',
@script=N'OutputDataSet<-InputDataSet',
@input_data_1 =N'select 1 as hello'
with result sets (([hello] int not null));
go
```

### For Python

```
exec sp_execute_external_script @language =N'Python',
@script=N'OutputDataSet= InputDataSet',
@input_data_1 =N'select 1 as hello'
with result sets (([hello] int not null));
go
```

## Errors generated by the extensibility framework

SQL Server generates separate logs for the external script language runtimes. These errors are not generated by the Python or R language. They're generated from the extensibility components in SQL Server, including language-specific launchers and their satellite processes.

You can get these logs from the following default locations:

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.<instance_name>\MSSQL\Log\ExtensibilityLog
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\MSSQL\Log\ExtensibilityLog
```

#### NOTE

The exact folder name differs based on the instance name. Depending on your configuration, the folder might be on a different drive.

For example, the following log messages are related to the extensibility framework:

- *LogonUser Failed for user MSSQLSERVER01*

This might indicate that the worker accounts that run external scripts cannot access the instance.

- *InitializePhysicalUsersPool Failed*

This message might mean that your security settings are preventing setup from creating the pool of worker accounts that are needed to run external scripts.

- *Security Context Manager initialization failed*
- *Satellite Session Manager initialization failed*

## System events

1. Open Windows Event Viewer, and search the **System Event** log for messages that include the string *Launchpad*.
2. Open the ExtLaunchErrorlog file, and look for the string *ErrorCode*. Review the message that's associated with the *ErrorCode*.

For example, the following messages are common system errors that are related to the SQL Server extensibility framework:

- *The SQL Server Launchpad (MSSQLSERVER) service failed to start due to the following error:*
- *The service did not respond to the start or control request in a timely fashion.*
- *A timeout was reached (120000 milliseconds) while waiting for the SQL Server Launchpad (MSSQLSERVER) service to connect.*

## Dump files

If you are knowledgeable about debugging, you can use the dump files to analyze a failure in Launchpad.

1. Locate the folder that contains the setup bootstrap logs for SQL Server. For example, in SQL Server 2016, the default path was C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log.
2. Open the bootstrap log subfolder that is specific to extensibility.
3. If you need to submit a support request, add the entire contents of this folder to a zipped file. For example, C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log\LOG\ExtensibilityLog.

The exact location might differ on your system, and it might be on a drive other than your C drive. Be sure to get the logs for the instance where machine learning is installed.

## Configuration settings

This section lists additional components or providers that can be a source of errors when you run R or Python scripts.

## **What network protocols are available?**

Machine Learning Services requires the following network protocols for internal communication among extensibility components, and for communication with external R or Python clients.

- Named pipes
- TCP/IP

Open SQL Server Configuration Manager to determine whether a protocol is installed and, if it is installed, to determine whether it is enabled.

## **Security configuration and permissions**

For worker accounts:

1. In Control Panel, open **Users and Groups**, and locate the group used to run external script jobs. By default, the group is **SQLRUserGroup**.
2. Verify that the group exists and that it contains at least one worker account.
3. In SQL Server Management Studio, select the instance where R or Python jobs will be run, select **Security**, and then determine whether there is a logon for SQLRUserGroup.
4. Review permissions for the user group.

For individual user accounts:

1. Determine whether the instance supports Mixed Mode authentication, SQL logins only, or Windows authentication only. This setting affects your R or Python code requirements.
2. For each user who needs to run R code, determine the required level of permissions on each database where objects will be written from R, data will be accessed, or objects will be created.
3. To enable script execution, create roles or add users to the following roles, as necessary:
  - All but *db\_owner*: Require EXECUTE ANY EXTERNAL SCRIPT.
  - *db\_datawriter*: To write results from R or Python.
  - *db\_ddladmin*: To create new objects.
  - *db\_datareader*: To read data that's used by R or Python code.
4. Note whether you changed any default startup accounts when you installed SQL Server 2016.
5. If a user needs to install new R packages or use R packages that were installed by other users, you might need to enable package management on the instance and then assign additional permissions. For more information, see [Enable or disable R package management](#).

## **What folders are subject to locking by antivirus software?**

Antivirus software can lock folders, which prevents both the setup of the machine learning features and successful script execution. Determine whether any folders in the SQL Server tree are subject to virus scanning.

However, when multiple services or features are installed on an instance, it can be difficult to enumerate all possible folders that are used by the instance. For example, when new features are added, the new folders must be identified and excluded.

Moreover, some features create new folders dynamically at runtime. For example, in-memory OLTP tables, stored procedures, and functions all create new directories at runtime. These folder names often contain GUIDs and cannot be predicted. The SQL Server Trusted Launchpad creates new working directories for R and Python script jobs.

Because it might not be possible to exclude all folders that are needed by the SQL Server process and its features, we recommend that you exclude the entire SQL Server instance directory tree.

## **Is the firewall open for SQL Server? Does the instance support remote connections?**

1. To determine whether SQL Server supports remote connections, see [Configure remote server connections](#).
2. Determine whether a firewall rule has been created for SQL Server. For security reasons, in a default installation, it might not be possible for remote R or Python client to connect to the instance. For more information, see [Troubleshooting connecting to SQL Server](#).

## See also

[Troubleshoot machine learning in SQL Server](#)

# Upgrade and installation FAQ for SQL Server Machine Learning or R Server

4/12/2018 • 6 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This topic provides answers to some common questions about installation of machine learning features in SQL Server. It also covers common questions about upgrades.

- Some problems occur only with upgrades from pre-release versions. Therefore, we recommend that you identify your version and edition first before reading these notes. To get version information, run `@@VERSION` in a query from SQL Server Management Studio.
- Upgrade to the most current release or service release as soon as possible, to resolve any issues that were fixed in recent releases.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services (In-Database)

## Requirements and restrictions on older versions of SQL Server 2016

Depending on the build of SQL Server that you are installing, some of the following limitations might apply:

- In early versions of SQL Server 2016 R Services, 8dot3 notation was required on the drive that contains the working directory. If you installed a pre-release version, upgrading to SQL Server 2016 Service Pack 1 should fix this issue. This requirement does not apply to releases after SP1.
- Side-by-side installation with another version of R, or with other releases from Revolution Analytics, is not supported.
- Disable virus scanning before beginning setup. After setup is completed, we recommend suspending virus scanning on the folders used by SQL Server. Preferably, suspend scanning on the entire SQL Server tree.
  - Installing Microsoft R Server on an instance of SQL Server installed on Windows Core. In the RTM version of SQL Server 2016, there was a known issue when adding Microsoft R Server to an instance on Windows Server Core edition. This has been fixed. If you encounter this issue, you can apply the fix described in [KB3164398](#) to add the R feature to the existing instance on Windows Server Core. For more information, see [Can't install Microsoft R Server Standalone on a Windows Server Core operating system](#).

## Offline installation of machine learning components for a localized version of SQL Server 2016

Early-release versions of SQL Server 2016 failed to install locale-specific .cab files during offline setup without an internet connection. This issue was fixed in later releases, but if the installer returns a message stating it cannot install the correct language, you can edit the filename to allow setup to continue.

- Manually edit the installer file to ensure that the correct language is installed. For example, to install the Japanese version of SQL Server, you would change the name of the file from `SRS_8.0.3.0_1033.cab` to `SRS_8.0.3.0_1041.cab`.
- The language identifier used for the machine learning components must be the same as the SQL Server

setup language, or you cannot complete setup.

## Pre-release versions: support policies, upgrade, and known issues

New installations of any pre-release version of R Services (In-Database) is no longer supported. If you are using a pre-release version, upgrade as soon as possible.

This section contains detailed instructions for specific upgrade scenarios.

### How to upgrade SQL Server

You can upgrade your version of SQL Server by re-running the setup wizard.

- [Upgrade SQL Server](#)
- [Upgrade SQL Server Using the Installation Wizard](#)

You can upgrade just the machine learning components by using a process called binding:

- [Use SqlBindR to upgrade machine learning components](#)

### End of support for in-place upgrades from prerelease versions

Upgrades from pre-release versions of SQL Server 2016 are no longer supported. This includes SQL Server 2016 CTP3, CTP3.1, CTP3.2, RC0, or RC1.

The following versions were installed with pre-release versions of SQL Server 2016.

VERSION	BUILD
CTP 3.0	13.0.xxx
CTP 3.1	13.0.801.12
CTP 3.2	13.0.900.73
CTP 3.3	13.0.1000.281
RC1	13.0.1200.242
RC2	13.0.1300.275
RC3	13.0.1400.361

If you have any doubt about which version you are using, run `@@VERSION` in a query from SQL Server Management Studio.

In general, the process for upgrade is as follows:

1. Back up scripts and data.
2. Uninstall the pre-release version.
3. Install a release version.

Uninstalling a pre-release version of the SQL Server machine learning components can be complex and could require running a special script. Contact technical support for assistance.

### Uninstall prior to upgrading from an older version of Microsoft R Server

If you installed a pre-release version of Microsoft R Server, you must uninstall it before you can upgrade to a newer version.

1. In **Control Panel**, click **Add/Remove Programs**, and select Microsoft SQL Server 2016 <version number>.
2. In the dialog box with options to **Add**, **Repair**, or **Remove** components, select **Remove**.
3. On the **Select Features** page, under **Shared Features**, select **R Server (Standalone)**. Click **Next**, and then click **Finish** to uninstall just the selected components.

## R Services and R Server (Standalone) side-by-side errors

In earlier versions of SQL Server 2016, installing both R Server (Standalone) and R Services (In-Database) at the same time sometimes caused setup to fail with an “access denied” message. This issue was fixed in Service Pack 1 for SQL Server 2016.

If you encountered this error, and need to upgrade these features, perform a slipstream installation of SQL Server 2016 with SP1. There are two ways to resolve the issue, both of which require uninstalling and reinstalling.

1. Uninstall R Services (In-Database), and make sure the user accounts for SQLRUserGroup are removed.
2. Restart the server, and then reinstall R Server (Standalone).
3. Run SQL Server setup once more, and this time select **Add Features to Existing SQL Server**.
4. Choose the instance, and then select the **R Services (In-Database)** option to add.

If this procedure fails to resolve the problem, try the following workaround:

1. Uninstall R Services (In-Database) and R Server (Standalone) at the same time.
2. Remove the local user accounts (SQLRUserGroup).
3. Restart the server.
4. Run SQL Server setup, and add the R Services (In-Database) feature only. Do not select **R Server (Standalone)**.

Generally, we recommend that you do not install both R Services (In-Database) and R Server (Standalone) on the same computer. However, assuming the server has sufficient capacity, you might find R Server Standalone might be useful as a development tool. Another possible scenario is that you need to use the operationalization features of R Server, but also want to access SQL Server data without data movement.

## Incompatible version of R Client and R Server

If you install Microsoft R Client and use it to run R in a remote SQL Server compute context, you might get an error like this:

*You are running version 9.0.0 of Microsoft R client on your computer, which is incompatible with the Microsoft R Server version 8.0.3. Download and install a compatible version.*

In SQL Server 2016, it was required that the version of R that was running in SQL Server R Services be exactly the same as the libraries in Microsoft R Client. That requirement has been removed in later versions. However, we recommend that you always get the latest versions of the machine learning components, and install all service packs.

If you have an earlier version of Microsoft R Server and need to ensure compatibility with Microsoft R Client 9.0.0, install the updates that are described in this [support article](#).

## Installation fails with error "Only one Revolution Enterprise product

## can be installed at a time."

You might encounter this error if you have an older installation of the Revolution Analytics products, or a pre-release version of SQL Server R Services. You must uninstall any previous versions before you can install a newer version of Microsoft R Server. Side-by-side installation with other versions of the Revolution Enterprise tools is not supported.

However, side-by-side installs are supported when using R Server Standalone with SQL Server 2017 (14.x) or SQL Server 2016.

## Registry cleanup to uninstall older components

If you have problems removing an older version, you might need to edit the registry to remove related keys.

### IMPORTANT

This issue applies only if you installed a pre-release version of Microsoft R Server or a CTP version of SQL Server 2016 R Services.

1. Open the Windows Registry, and locate this key: `HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall`.
2. Delete any of the following entries if present, and if the key contains only the value `sEstimatedSize2`:
  - E0B2C29E-B8FC-490B-A043-2CAE75634972 (for 8.0.2)
  - 46695879-954E-4072-9D32-1CC84D4158F4 (for 8.0.1)
  - 2DF16DF8-A2DB-4EC6-808B-CB5A302DA91B (for 8.0.0)
  - 5A2A1571-B8CD-4AAF-9303-8DF463DABE5A (for 7.5.0)

## See also

[SQL Server Machine Learning Services \(In-Database\)](#)

[SQL Server Machine Learning Server \(Standalone\)](#)

# Common issues with Launchpad service and external script execution in SQL Server

6/1/2018 • 10 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

SQL Server Trusted Launchpad service supports external script execution for R and Python. On SQL Server 2016 R Services, SP1 provides the service. SQL Server 2017 includes the Launchpad service as part of the initial installation.

Multiple issues can prevent Launchpad from starting, including configuration problems or changes, or missing network protocols. This article provides troubleshooting guidance for many issues. For any we missed, you can post questions to the [Machine Learning Server forum](#).

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Determine whether Launchpad is running

1. Open the **Services** panel (Services.msc). Or, from the command line, type **SQLServerManager13.msc** or **SQLServerManager14.msc** to open [SQL Server Configuration Manager](#).
2. Make a note of the service account that Launchpad is running under. Each instance where R or Python is enabled should have its own instance of the Launchpad service. For example, the service for a named instance might be something like *MSSQLLaunchpad\$instanceName*.
3. If the service is stopped, restart it. On restarting, if there are any issues with configuration, a message is published in the system event log, and the service is stopped again. Check the system event log for details about why the service stopped.
4. Review the contents of RSetup.log, and make sure that there are no errors in the setup. For example, the message *Exiting with code 0* indicates failure of the service to start.
5. To look for other errors, review the contents of rlauncher.log.

## Check the Launchpad service account

The default service account might be "NT Service\$SQL2016" or "NT Service\$SQL2017". The final part might vary, depending on your SQL instance name.

The Launchpad service (Launchpad.exe) runs by using a low-privilege service account. However, to start R and Python and communicate with the database instance, the Launchpad service account requires the following user rights:

- Log on as a service (SeServiceLogonRight)
- Replace a process-level token (SeAssignPrimaryTokenPrivilege)
- Bypass traverse checking (SeChangeNotifyPrivilege)
- Adjust memory quotas for a process (SeIncreaseQuotaSizePrivilege)

For information about these user rights, see the "Windows privileges and rights" section in [Configure Windows service accounts and permissions](#).

**TIP**

If you are familiar with the use of the Support Diagnostics Platform (SDP) tool for SQL Server diagnostics, you can use SDP to review the output file with the name MachineName\_UserRights.txt.

## User group for Launchpad cannot log on locally

During setup of Machine Learning services, SQL Server creates the Windows user group **SQLRUUserGroup** and then provisions it with all rights necessary for Launchpad to connect to SQL Server and run external script jobs. If this user group is enabled, it is also used to execute Python scripts.

However, in organizations where more restrictive security policies are enforced, the rights that are required by this group might have been manually removed, or they might be automatically revoked by policy. If the rights have been removed, Launchpad can no longer connect to SQL Server, and SQL Server cannot call the external runtime.

To correct the problem, ensure that the group **SQLRUUserGroup** has the system right **Allow log on locally**.

For more information, see [Configure Windows service accounts and permissions](#).

## Permissions to run external scripts

Even if Launchpad is configured correctly, it returns an error if the user does not have permission to run R or Python scripts.

If you installed SQL Server as a database administrator or you are a database owner, you are automatically granted this permission. However, other users usually have more limited permissions. If they try to run an R script, they get a Launchpad error.

To correct the problem, in SQL Server Management Studio, a security administrator can modify the SQL login or Windows user account by running the following script:

```
GRANT EXECUTE ANY EXTERNAL SCRIPT TO <username>
```

For more information, see [GRANT \(Transact-SQL\)](#).

## Common Launchpad errors

This section lists the most common error messages that Launchpad returns.

### "Unable to launch runtime for R script"

If the Windows group for R users (also used for Python) cannot log on to the instance that is running R Services, you might see the following errors:

- Errors generated when you try to run R scripts:
  - *Unable to launch runtime for 'R' script. Please check the configuration of the 'R' runtime.*
  - *An external script error occurred. Unable to launch the runtime.*
- Errors generated by the SQL Server Trusted Launchpad service:
  - *Failed to initialize the launcher RLauncher.dll*
  - *No launcher dlls were registered!*
  - *Security logs indicate that the account NT SERVICE was unable to log on*

For information about how to grant this user group the necessary permissions, see [Install SQL Server 2016 R Services](#).

**NOTE**

This limitation does not apply if you use SQL logins to run R scripts from a remote workstation.

## "Logon failure: the user has not been granted the requested logon type"

By default, SQL Server Trusted Launchpad uses the following account on startup: `NT Service\MSSQLLaunchpad`. The account is configured by SQL Server setup to have all necessary permissions.

If you assign a different account to Launchpad, or the right is removed by a policy on the SQL Server machine, the account might not have the necessary permissions, and you might see this error:

*ERROR\_LOGON\_TYPE\_NOT\_GRANTED 1385 (0x569) Logon failure: the user has not been granted the requested logon type at this computer.*

To grant the necessary permissions to the new service account, use the Local Security Policy application, and update the permissions on the account to include the following permissions:

- Adjust memory quotas for a process (SeIncreaseQuotaPrivilege)
- Bypass traverse checking (SeChangeNotifyPrivilege)
- Log on as a service (SeServiceLogonRight)
- Replace a process-level token (SeAssignPrimaryTokenPrivilege)

## "Unable to communicate with the Launchpad service"

If you have installed and then enabled machine learning, but you get this error when you try to run an R or Python script, the Launchpad service for the instance might have stopped running.

1. From a Windows command prompt, open the SQL Server Configuration Manager. For more information, see [SQL Server Configuration Manager](#).
2. Right-click SQL Server Launchpad for the instance, and then select **Properties**.
3. Select the **Service** tab, and then verify that the service is running. If it is not running, change the **Start Mode** to **Automatic**, and then select **Apply**.
4. Restarting the service usually fixes the problem so that machine learning scripts can run. If the restart does not fix the issue, note the path and the arguments in the **Binary Path** property, and do the following:
  - a. Review the launcher's .config file and ensure that the working directory is valid.
  - b. Ensure that the Windows group that's used by Launchpad can connect to the SQL Server instance, as described in the [previous section](#).
  - c. If you change any of the service properties, restart the Launchpad service.

## "Fatal error creation of tmpFile failed"

In this scenario, you have successfully installed machine learning features, and Launchpad is running. You try to run some simple R or Python code, but Launchpad fails with an error like the following:

*Unable to communicate with the runtime for R script. Please check the requirements of R runtime.*

At the same time, the external script runtime writes the following message as part of the STDERR message:

*Fatal error: creation of tmpfile failed.*

This error indicates that the account that Launchpad is attempting to use does not have permission to log on to the database. This situation can happen when strict security policies are implemented. To determine whether this is the case, review the SQL Server logs, and check to see whether the MSSQLSERVER01 account was denied at login. The same information is provided in the logs that are specific to R\_SERVICES or PYTHON\_SERVICES. Look for ExtLaunchError.log.

By default, 20 accounts are set up and associated with the Launchpad.exe process, with the names MSSQLSERVER01 through MSSQLSERVER20. If you make heavy use of R or Python, you can increase the number of accounts.

To resolve the issue, ensure that the group has *Allow Log on Locally* permissions to the local instance where machine learning features have been installed and enabled. In some environments, this permission level might require a GPO exception from the network administrator.

## "Not enough quota to process this command"

This error can mean one of several things:

- Launchpad might have insufficient external users to run the external query. For example, if you are running more than 20 external queries concurrently, and there are only 20 default users, one or more queries might fail.
- Insufficient memory is available to process the R task. This error happens most often in a default environment, where SQL Server might be using up to 70 percent of the computer's resources. For information about how to modify the server configuration to support greater use of resources by R, see [Operationalizing your R code](#).

## "Can't find package"

If you run R code in SQL Server and get this message, but did not get the message when you ran the same code outside SQL Server, it means that the package was not installed to the default library location used by SQL Server.

This error can happen in many ways:

- You installed a new package on the server, but access was denied, so R installed the package to a user library.
- You installed R Services and then installed another R tool or set of libraries, including Microsoft R Server (Standalone), Microsoft R Client, RStudio, and so forth.

To determine the location of the R package library that's used by the instance, open SQL Server Management Studio (or any other database query tool), connect to the instance, and then run the following stored procedure:

```
EXEC sp_execute_external_script @language = N'R',
@script = N' print(normalizePath(R.home())); print(.libPaths());'
```

### Sample results

*STDOUT message(s) from external script:*

```
[1] "C:\Program Files\Microsoft SQL Server\MSSQL13.SQL2016\R_SERVICES"
```

```
[1] "C:/Program Files/Microsoft SQL Server/MSSQL13.SQL2016/R_SERVICES/library"
```

To resolve the issue, you must reinstall the package to the SQL Server instance library.

**NOTE**

If you have upgraded an instance of SQL Server 2016 to use the latest version of Microsoft R, the default library location is different. For more information, see [Use SqlBindR to upgrade an instance of R Services](#).

## Launchpad shuts down due to mismatched DLLs

If you install the database engine with other features, patch the server, and then later add the Machine Learning feature by using the original media, the wrong version of the Machine Learning components might be installed. When Launchpad detects a version mismatch, it shuts down and creates a dump file.

To avoid this problem, be sure to install any new features at the same patch level as the server instance.

**The wrong way to upgrade:**

1. Install SQL Server 2016 without R Services.
2. Upgrade SQL Server 2016 Cumulative Update 2.
3. Install R Services (In-Database) by using the RTM media.

**The correct way to upgrade:**

1. Install SQL Server 2016 without R Services.
2. Upgrade SQL Server 2016 to the desired patch level. For example, install Service Pack 1 and then Cumulative Update 2.
3. To add the feature at the correct patch level, run SP1 and CU2 setup again, and then choose R Services (In-Database).

## Launchpad fails to start if 8dot3 notation is required

**NOTE**

On older systems, Launchpad can fail to start if there is an 8dot3 notation requirement. This requirement has been removed in later releases. SQL Server 2016 R Services customers should install one of the following:

- SQL Server 2016 SP1 and CU1: [Cumulative Update 1 for SQL Server](#).
- SQL Server 2016 RTM, Cumulative Update 3, and this [hotfix](#), which is available on demand.

For compatibility with R, SQL Server 2016 R Services (In-Database) required the drive where the feature is installed to support the creation of short file names by using *8dot3 notation*. An 8.3 file name is also called a *short file name*, and it's used for compatibility with earlier versions of Microsoft Windows or as an alternative to long file names.

If the volume where you are installing R does not support short file names, the processes that launch R from SQL Server might not be able to locate the correct executable, and Launchpad will not start.

As a workaround, you can enable the 8dot3 notation on the volume where SQL Server is installed and where R Services is installed. You must then provide the short name for the working directory in the R Services configuration file.

1. To enable 8dot3 notation, run the fsutil utility with the *8dot3name* argument as described here: [fsutil 8dot3name](#).

2. After the 8dot3 notation is enabled, open the RLauncher.config file and note the property of `WORKING_DIRECTORY`. For information about how to find this file, see [Data collection for Machine Learning troubleshooting](#).
3. Use the fsutil utility with the *file* argument to specify a short file path for the folder that's specified in `WORKING_DIRECTORY`.
4. Edit the configuration file to specify the same working directory that you entered in the `WORKING_DIRECTORY` property. Alternatively, you can specify a different working directory and choose an existing path that's already compatible with the 8dot3 notation.

## Next steps

[Machine Learning Services troubleshooting and known issues](#)

[Data collection for troubleshooting machine learning](#)

[Upgrade and installation FAQ](#)

[Troubleshoot database engine connections](#)

# R scripting errors in SQL Server

6/1/2018 • 4 minutes to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (Windows only) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

This article documents several scriptin gerrors when running R code in SQL Server. The list is not comprehensive. There are many packages, and errors can vary between versions of the same package. We recommend posting script errors on the [Machine Learning Server forum](#), which supports the machine learning components used in R Services (In-Database), Microsoft R Client, and Microsoft R Server.

**Applies to:** SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services

## Valid script fails in T-SQL or in stored procedures

Before wrapping your R code in a stored procedure, it is a good idea to run your R code in an external IDE, or in one of the R tools such as RTerm or RGui. By using these methods, you can test and debug the code by using the detailed error messages that are returned by R.

However, sometimes code that works perfectly in an external IDE or utility might fail to run in a stored procedure or in a SQL Server compute context. If this happens, there are a variety of issues to look for before you can assume that the package doesn't work in SQL Server.

1. Check to see whether Launchpad is running.
2. Review messages to see whether either the input data or output data contains columns with incompatible or unsupported data types. For example, queries on a SQL database often return GUIDs or RowGUIDs, both of which are unsupported. For more information, see [R libraries and data types](#).
3. Review the help pages for individual R functions to determine whether all parameters are supported for the SQL Server compute context. For ScaleR help, use the inline R help commands, or see [Package Reference](#).

If the R runtime is functioning but your script returns errors, we recommend that you try debugging the script in a dedicated R development environment, such as R Tools for Visual Studio.

We also recommend that you review and slightly rewrite the script to correct any problems with data types that might arise when you move data between R and the database engine. For more information, see [R libraries and data types](#).

Additionally, you can use the sqlrutils package to bundle your R script in a format that is more easily consumed as a stored procedure. For more information, see:

- [Generate a stored procedure for R code by using the sqlrutils package](#)
- [Create a stored procedure by using sqlrutils](#)

## Script returns inconsistent results

R scripts can return different values in a SQL Server context, for several reasons:

- Implicit type conversion is automatically performed on some data types, when the data is passed between SQL Server and R. For more information, see [R libraries and data types](#).
- Determine whether bitness is a factor. For example, there are often differences in the results of math operations for 32-bit and 64-bit floating point libraries.

- Determine whether NaNs were produced in any operation. This can invalidate results.
- Small differences can be amplified when you take a reciprocal of a number near zero.
- Accumulated rounding errors can cause such things as values that are less than zero instead of zero.

## Implied authentication for remote execution via ODBC

If you connect to the SQL Server computer to run R commands by using the **RevoScaleR** functions, you might get an error when you use ODBC calls that write data to the server. This error happens only when you're using Windows authentication.

The reason is that the worker accounts that are created for R Services do not have permission to connect to the server. Therefore, ODBC calls cannot be executed on your behalf. The problem does not occur with SQL logins because, with SQL logins, the credentials are passed explicitly from the R client to the SQL Server instance and then to ODBC. However, using SQL logins is also less secure than using Windows authentication.

To enable your Windows credentials to be passed securely from a script that's initiated remotely, SQL Server must emulate your credentials. This process is termed *implied authentication*. To make this work, the worker accounts that run R or Python scripts on the SQL Server computer must have the correct permissions.

1. Open SQL Server Management Studio as an administrator on the instance where you want to run R code.
2. Run the following script. Be sure to edit the user group name, if you changed the default, and the computer and instance names.

```
USE [master]
GO

CREATE LOGIN [computername\SQLRUserGroup] FROM WINDOWS WITH
DEFAULT_DATABASE=[master], DEFAULT_LANGUAGE=[language]
GO
```

## Avoid clearing the workspace while you're running R in a SQL compute context

Although clearing the workspace is common when you work in the R console, it can have unintended consequences in a SQL compute context.

`revoScriptConnection` is an object in the R workspace that contains information about an R session that's called from SQL Server. However, if your R code includes a command to clear the workspace (such as `rm(list=ls())`), all information about the session and other objects in the R workspace is cleared as well.

As a workaround, avoid indiscriminate clearing of variables and other objects while you're running R in SQL Server. You can delete specific variables by using the **remove** function:

```
remove('name1', 'name2', ...)
```

If there are multiple variables to delete, we suggest that you save the names of temporary variables to a list and then perform periodic garbage collections on the list.

## Next steps

[Machine Learning Services troubleshooting and known issues](#)

[Data collection for troubleshooting machine learning](#)

[Upgrade and installation FAQ](#)

[Troubleshoot database engine connections](#)