Table of Contents

Microsoft Machine Learning Services

6/1/2017 • 4 min to read • Edit Online

The goal of Microsoft Machine Learning Services is to provide an extensible, scalable platform for integrating machine learning tasks and tools with the applications that consume machine learning services. The platform must serve the needs of all users involved in the data development and analytics process, from data scientists, to architects and database administrators.

Key concepts include:

- High-performance scalable analytics. Write-once, deploy-anywhere solutions
- Avoid data movement and data risk: bring analytics to the data
- Let data scientists choose their own tools and languages
- Integrate the best features of open source with Microsoft's enterprise capabilities
- Simplify administration
- Make it easier to deploy and consume predictive models

Roadmap

In SQL Server 2016, Microsoft launched two server platforms for integrating the popular open source R language with business applications:

- **SQL Server R Services (In-Database)**, for integration with SQL Server
- **Microsoft R Server**, for enterprise-level R deployments on Windows and Linux servers

In SQL Server 2017 CTP 2.0, the name has been changed to reflect support for the popular Python language.

- **SQL Server Machine Learning Services (In-Database)** supports both R and Python. version-md.md)]
- **Microsoft Machine Learning Server** supports R and Python deployments on Windows servers, with expansion to other supported platforms planned for late 2017.

In-Database Analytics with SQL Server

R Services (In-Database) brings the compute to the data by allowing R to run on the same computer as the database. It includes the Lacunhpad service, which runs outside the SQL Server process and communicates securely with the R or Python runtimes.

Using SQL Server Machine Learning Services (or SQL Server R Services), you can train models, generate plots, perform scoring, and easily move data between SQL Server and R or Python.

Data scientists who are testing and developing solutions can communicate with the server from a remote development computer to run code on the server, or deploy completed solutions to SQL Server by embedding machine learnign scripts in stored procedures.

When you install machine learning services for SQL Server, you get a distribution of the open source R language, as well as the Microsoft R tools and libraries, which include high-performance, scalable R and Python packages, as well as SQL Server services for faster connectivity with R or Python runtimes.

To get started, see SQL Server R Services. For walkthroughs and tutorials, see Machine Learning Tutorials.

NOTE

Support for Python is provided only in SQL Server 2017. This is a new feature and still under development.

Microsoft R Server and Machine Learning Server (Standalone)

This standalone server system supports distributed, scalable R solutions on multiple platforms and using multiple enterprise data sources, including Linux, Hadoop, and Teradata.

For more information, see Microsoft R Server (MSDN).

If you don't need to integrate with SQL Server, you can install R Server to enable rapid development, deployment, and operationalization of machine learning solutions.

If you install Microsoft Machine Learning Server using SQL Server 2017 setup, you can also deploy and consume Python applications.

Related Technologies

Microsoft provides broad support for machine learning ecosystems, including tools, providers, enhanced R and Python packages, a cloud development and services platform, and an integrated development environment.

R Tools for Visual Studio

Visual Studio provides a full development environment for the R language. The plug-in includes an editor, interactive window, plotting, debugger, and more. You can use .NET languages from R or invoke R from .NET via open source libraries such as R.NET and rClr.

Visual Studio also has an excellent Python development environment. There's no easier way to work with machine learning languages while continuing to enjoy access to SQL database development tools.

For more information, see:

- R Tools for Visual Studio
- Python - Visual Studio

Azure Machine Learning

When you create your own workspace in Azure Machine Learning Studio, you'll have access to over 400 preloaded R packages. You can also choose when you create an experiment that uses R, to deploy R using either a standard CRAN R distribution, or Microsoft R Open. You can even create your own R packages and upload them to Azure to run as custom modules.

For more information, see:

- Extend your experiment with R
- Author custom R modules in Azure Machine Learning

Many of the algorithms provided in Azure ML are now included in Microsoft Machine Learning Services, as part of the MicrosoftML package. For more information, see MicrosoftML on MSDN.

Azure Machine Learning is also a convenient platform if you want to build and train models to deploy as a Web service, and publish solutions to the Machine Learning Marketplace.

Data Science Virtual Machines

You can deploy a pre-installed and pre-configured version of Microsoft R Server in Microsoft Azure, making it easy to get started with data exploration and modeling right away on the cloud without setting up a fully configured system on premises.

The Azure Marketplace contains several virtual machines that support data science:

- The **Microsoft Data Science Virtual Machine** is configured with Microsoft R Server, as well as Python (Anaconda distribution), a Jupyter notebook server, Visual Studio Community Edition, Power BI Desktop, the Azure SDK, and SQL Server Express edition.
- **Microsoft R Server 2016 for Linux** contains the latest version of R Server (version 9.0.1). Separate VMs are available for CentOS version 7.2 and Ubuntu version 16.04.
- The **R Server Only SQL Server 2016 Enterprise** virtual machine includes a standalone installer for R Server 9.0.1 that supports the new Modern Software Lifecycle licensing model.

See Also

Getting Started with SQL Server R Services

Getting started with Microsoft R Server

Install SQL Server Database Engine

Getting Started with Machine Learning

6/12/2017 • 2 min to read • Edit Online

Microsoft provides an integrated, scalable set of machine learning solutions for both on-premises and the cloud:

- **Integrated**, because you can run R or Python in SQL Server. This lets you easily merge enterprise work flows for ETL and reporting with data science tasks such as feature engineering, model creation, and scoring.
- **Scalable**, because the data scientist can develop and test a solution on a laptop, and then deploy it to multiple platforms for distributed or parallel processing of key operations such as model training and scoring. Supported platforms include SQL Server on Windows, Hadoop, and Spark.

This article provides links to resources for each product in the Microsoft Machine Learning platform.

Products and Platforms

SQL Server 2016

SQL Server 2016 supports running R code in SQL Server, or using the SQL Server as the remote compute context for R jobs. This provides security for your data and lets you manage and balance resources used by R.

SQL Server 2017

In SQL Server 2017 CTP 2.0, support for Python was added, and the name was changed to Machine Learning Services (In-Database).

- Set up SQL Server R Services or Machine Learning Services

Microsoft R Server

If you don't need to use SQL Server data, install Microsoft R Server to enjoy distributed, scalable R processing. You can install R Server either through , or through SQL Server setup.

- Introducing Microsoft R Server

Microsoft Machine Learning Server

If you want to try out Python integration, be sure to install the latest version, **Machine Learning Server**, which is available only through SQL Server 2017 setup: Set up Microsoft R Server or Machine Learning Server

Related Products

Use this standalone installer if you want to set up a server for machine learning but don't need SQL Server

- Microsoft R Server

If you already installed one of the machine learning server products, this article provides information about how to set up a separate computer for development and testing of solutions, including tools and required libraries.

- Set Up a Data Science Client

Try out the complete machine learning solution on this virtual machine, which includes the server and all development tools, a Python Anaconda distribution, Jupyter notebooks, and the latest version of Microsoft R.

- Data Science Virtual Machine

Demos, Blogs, and Tutorials

These demos illustrate how a ski rental business might use machine learning to predict future rentals and schedule staff to meet demand.

- Build an intelligent app with SQL Server and R
- Build an intelligent app with SQL Server and Python

See this page for links to tutorials and end-to-end solutions for using R in SQL Server.

Current examples also include how to run Python in SQL Server, or build a model using Python and use it to score SQL Server data.

- Machine Learning Tutorials

Need to deploy a model to a web service using Machine Learning Server? See this walkthrough, complete with code:

- Publish and Consume Python Code

New to R? Wondering what RevoScaleR offers that R doesn't? See this R Server quick-start.

- Explore R and Scale R in 25 Short Functions

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, and includes sample data, and customizable code.

- Solution templates

See Also

Get Started with SQL Server R Services

Get Started with Microsoft R Server

What's New in Machine Learning Services in SQL Server

4/29/2017 • 3 min to read • Edit Online

In SQL Server 2016, Microsoft introduced SQL Server R Services, a feature that supports enterprise-scale data science, by integrating the R language with SQL Server database engine.

In SQL Server 2017, machine learning becomes even more powerful, with addition of support for the popular Python language. To reflect the support for multiple languages, as of CTP 2.0, SQL Server R Services has also been renamed as **Machine Learning Services (In-Database)**.

What's New in SQL Server 2017 CTP 2.0

SQL Server 2017 includes many new features to make it easier to build and deploy machine learning solutions that use SQL Server data.

> **IMPORTANT**
>
> Machine learning services, including use of R or Python, are currently not supported when running SQL Server on Linux. This feature will be added in a later release.

Python integration in SQL Server

You can now run Python in SQL Server, using stored procedures or remote compute contexts.

Machine learning with Python includes the **revoscalepy** module, which supports a subset of the distributed algorithms and compute contexts provided in RevoScaleR. More algorithms and transformations will be added soon.

For more information, see these topics:

- Set up Python in Machine Learning Services

"Pleasingly parallel" compute for distributed models

Often the data scientist needs to efficiently train a large number of small models over subsets of a very large dataset. For example, a large dataset might include data on multiple products or devices over many days, but thee data scientist or business decision maker needs models that are specific to a certain product, device, user base, or time period.

You can easily create multiple models in parallel from R, by using the new **rxExecBy** function. The function accepts a dataset containing ungrouped and unordered data, and lets you partition it by a single entity for training and model building. For example, you could partition by product, and then process the entire dataset in parallel. The outcome is multiple models, each model trained on the appropriate subset of data for each product.

The rxExecBy function is available in SQL Server 2017 CTP 2.0 and in Microsoft R Server 9.1.0. Supported compute contexts include RxSpark and RxInSQLServer.

For more information, see Create Multiple Models using rxExecBy.

Other Updates

This section lists features that were released previously, but that have been updated in CTP 2.0.

Improved package management for data scientists

SQL Server now provides roles for managing permissions associated with packages. The DBA can control packages at the database level and assign users the right to install their own packages, or share packages with other users. Users who belong to these roles can install and uninstall R packages on the SQL Server computer from a remote development client, without having to go through the database administrator each time.

In CTP 2.0, new functions have been added that let you easily back up and restore the package collections associated with users when you move or restore a database.

For more information, see R Package Management for SQL Server R Services.

Upgrade your R experience

The RevoScaleR package is included in SQL Server 2016, SQL Server 2017, and Microsoft R Server. It includes transforms and algorithms that support distributed or parallel processing, and multiple compute contexts.

If you installed an earlier version of RevoScaleR with SQL Server 2016, you can now upgrade to the very latest version by switching your server to use the Modern Lifecycle policy. By doing so you can take advantage of a faster release cycle for R and automatically upgrade all R components. For more information, see Microsoft R Server 9.0.1.

In CTP 2.0, Microsoft R components have been upgraded to version 9.1.0.

New functions and features in MicrosoftML

MicrosoftML is a machine learning package for R, created by the Microsoft Data Science team. MicrosoftML brings increased speed, performance and scale for handling a large corpus of text data, and high-dimensional categorical data in R models, with just a few lines of code. It also includes five fast, highly accurate learners that are included in Azure Machine Learning.

In CTP 2.0, MicrosoftML includes new image and test featurization functions, as well as support for parallelizable models with rxExecby.

For more information, see these topics:

- Introduction to MicrosoftML
- Overview of MicrosoftML Functions
- Reference.

Architecture and Overview of Machine Learning Services

6/1/2017 • 4 min to read • Edit Online

This topic 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 topics:

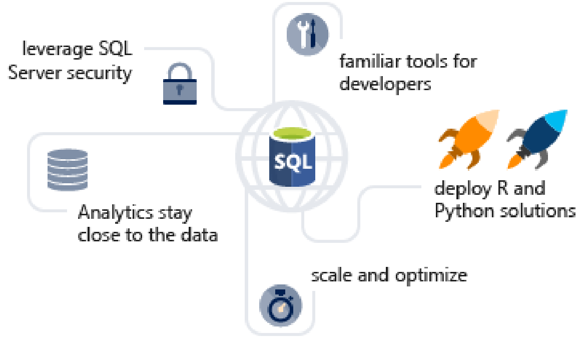
- Architecture Overview for SQL Server R Services
- Architecture Overview for Python in SQL Server

Design Goals

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.


leverage SQL Server security
familiar tools for developers
SQL
deploy R and Python solutions
Analytics stay close to the data
scale and optimize

- 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 Pillars

To meet these goals, the architecture of SQL Server 2016 R Services and SQL Server 2017 Machine 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 DLLs 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.

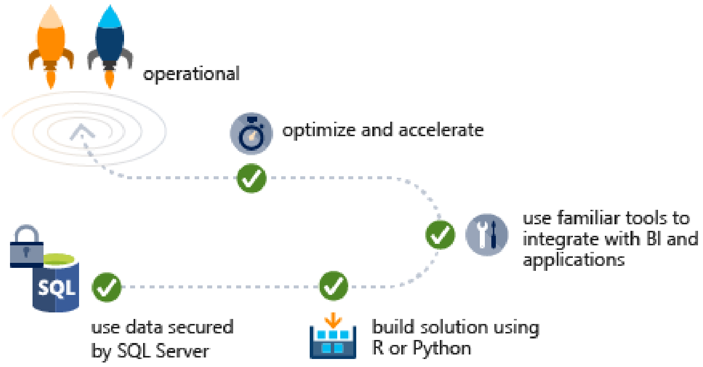
- **Scalability and performance** Integration with SQL Server is key to improving the usefulness of R and Python in the enterprise. Any R or Python script can be run by calling a stored procedure, and the results are returned as tabular results directly to SQL Server, making it easy to generate or consume 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 Workflow

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.


operational
optimize and accelerate
use familiar tools to integrate with BI and applications
SQL
use data secured by SQL Server
build solution using R or Python

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
- Data Science Virtual Machine
- Azure Cognitive Services
- Azure Machine Learning

See Also

R Server Standalone

Differences in R Features between Editions of SQL Server

6/12/2017 • 3 min to read • Edit Online

Support for machine learning is available in the following editions of SQL Server 2016 and SQL Server 2017.

- **Enterprise Edition**

  Includes R Services, for in-database analytics in SQL Server. Also includes R Server (Standalone), which can be used to connect to a variety of databases and pull data for analysis at scale, but does not run in-database. In SQL Server 2017, the equivalent features are Machine Learning Services (In-Database) and Machine Learning Server (Standalone).

  No restrictions. Optimized performance and scalability through parallelization and streaming. Supports analysis of large datasets that do not fit in the available memory, by using enhanced R packages, streaming, and parallel execution.

  Newer editions of Microsoft R Server include an improved version of the operationalization engine (formerly known as DeployR) that supports rapid, secure deployment and sharing of R solutions. For more information, see Operationalize.

  In-database analytics in SQL Server supports resource governance of external scripts to customize server resource usage.

- **Developer Edition**

  Same capabilities as Enterprise Edition; however, Developer Edition cannot be used in production environments.

- **Standard Edition**

  Has all the capabilities of in-database analytics included with Enterprise Edition, except for resource governance. Performance and scale is also limited: the data that can be processed must fit in server memory, and processing is limited to a single compute thread, even when using the **RevoScaleR** functions.

- **Express And Web Editions**

  Only Express Edition with Advanced Services includes the machine learning features. The performance limitations are similar to Standard Edition.

For more information about other product features, see Editions and Supported Features for SQL Server 2016

Enterprise Edition

Performance of machine learning solutions in SQL Server is expected to generally be better than any conventional implementation using 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. Performance has not been assessed for Python solutions, as the feature is still under development, but some of the same benefits are expected to apply.

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, streaming, and increased threads available for R worker processing.

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.

Developer Edition

Developer Edition provides performance equivalent to that of Enterprise Edition; however, use of Developer Edition is not supported for production environments.

Standard Edition

Even Standard Edition should offer some performance benefit, in comparison to standard R packages, given the same hardware configuration.

However, Standard Edition does not support Resource Governor. Using resource governance is the best way to customize server resources to support varied workloads such as model training and scoring.

Standard Edition also provides limited performance and scalability in comparison to Enterprise and Developer Editions. All the **ScaleR** functions and packages are included with Standard Edition, but the service that launches and manages R scripts is limited in the number of processes it can use. Moreover, data processed by the script must fit in memory. The same restrictions apply to solutions that use **revoscalepy**.

Express Edition with Advanced Services

Express Edition is subject to the same limitations as Standard Edition.

All Editions

The following machine learning languages are supported for all editions:

- SQL Server 2016: R
- SQL Server 2017: R and Python

Microsoft R Open is included with all editions.

Microsoft R Client can work with all editions.

See Also

Editions and Supported Features for SQL Server 2016

SQL Server R Services

4/19/2017 • 2 min to read • Edit Online

R Services (In-Database) provides a platform for developing and deploying intelligent applications that uncover new insights. You can use the rich and powerful R language and the many packages from the community to create models and generate predictions using your SQL Server data. Because R Services (In-Database) integrates the R language with SQL Server, you can keep analytics close to the data and eliminate the costs and security risks associated with data movement.

R Services (In-Database) supports the open source R language with a comprehensive set of SQL Server tools and technologies that offer superior performance, security, reliability and manageability. You can deploy R solutions using convenient, familiar tools, and your production applications can call the R runtime and retrieve predictions and visuals using Transact-SQL. You also get the ScaleR libraries to improve the scale and performance of your R solutions.

Through SQL Server setup, you can install both server and client components.

- **R Services (In-Database):** Install this feature during SQL Server setup to enable secure execution of R scripts on the SQL Server computer.

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

- **Microsoft R Server (Standalone):** A distribution of open source R combined with proprietary packages that support parallel processing and other performance improvements. Both R Services (In-Database) and Microsoft R Server (Standalone) include the base R runtime and packages, plus the **ScaleR** libraries for enhanced connectivity and performance.

- Microsoft R Client is available as a separate, free installer. You can use Microsoft R Client to develop solutions that can be deployed to R Services running on SQL Server, or to Microsoft R Server running on Windows, Teradata, or Hadoop.

> **NOTE**
>
> If you need to run your R code in SQL Server, be sure to install R Services (In-Database) as described here.
>
> Microsoft R Server (Standalone) is a separate option designed for using the ScaleR libraries on a Windows computer that is not running SQL Server.
>
> However, if you have Enterprise Edition, we recommend that you install Microsoft R Server (Standalone) on a laptop or other computer used for R development, to create R solutions that can easily be deployed to an instance of SQL Server that is running R Services (In-Database).

Additional Resources

Getting Started with SQL Server R Services
Describes common scenarios for uses of R with SQL Server.

Set Up SQL Server R Services In-Database
Install R and associated database components as part of SQL Server setup.

SQL Server R Services Tutorials

Learn how to create SQL Server data sources in your R code, and how to use remote compute contexts. Other tutorials aimed at SQL developers demonstrate how to train and deploy an R model in SQL Server.

See Also

Getting Started with Microsoft R Server (Standalone)

Set up a Standalone R Server)

Getting Started with SQL Server Machine Learning

5/17/2017 • 5 min to read • Edit Online

Machine Learning Services in SQL Server is designed to support data science tasks without exposing your data to security risks or moving data unnecesarily.

- In SQL Server 2016, you can work with your favorite R tools, but scale your analysis to billions of records while boosting performance. Integrating machine learning with SQL Server also means that you can put R code into production without having to re-code.
- SQL Server 2017 adds support for Python code, using the same extensibility framework.

This topic describes the key scenarios for using R or Python in-database, and provides resources to help you get started with your own solutions.

Applies to: SQL Server 2016 R Services, SQL Server 2017 Machine Learning Services (In-Database)

> **NOTE**
>
> SQL Server 2016 includes support only for the R language. Support for Python language requires SQL Server 2017 CTP 2.0.

Step 1. Set Up SQL Server Machine Learning Services

1. Run SQL Server setup and install at least one instance of the SQL Server database engine.
2. Add the feature that supports execution of external runtimes.
3. In SQL Server 2016, R is added by default. In SQL Server 2017, you must select a language to add. You can select either R or Python, or enable both.
4. When setup is done, perform some additional steps to enable external script execution, and restart the server.

Resources

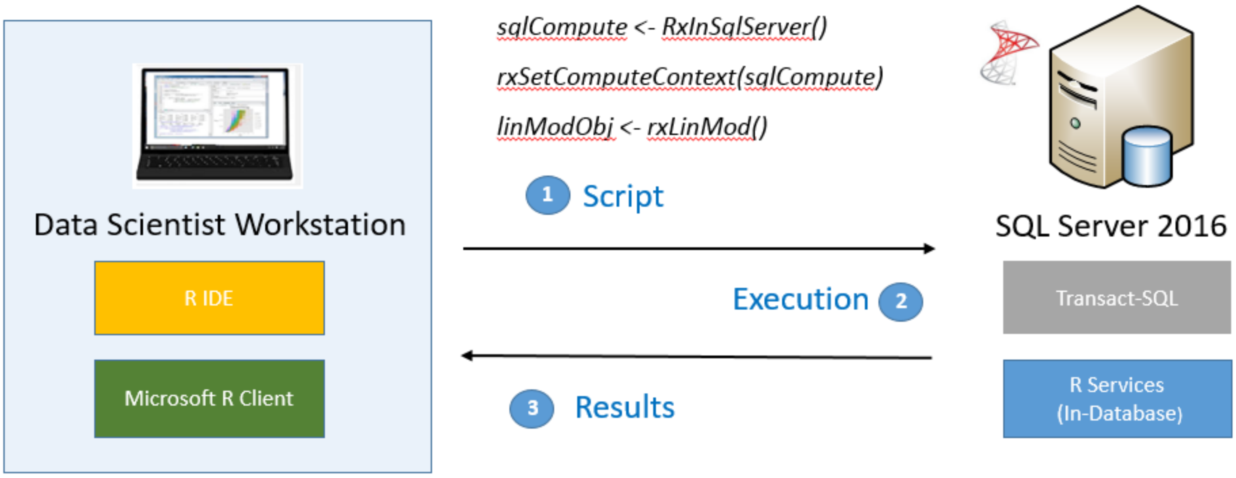
- Set up SQL Server with Machine Learning

> **TIP**
>
> Need to create a server for R jobs but don't need SQL Server? Try Microsoft R Server.

Step 2. Develop Your R or Python Solutions

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 Machine Learning Services 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 **ScaleR** 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.

Resources

- Install R Tools for Visual Studio or RStudio.

Step 3. Optimize

When the model is ready to scale on enterprise data, the data scientist will often work 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, new and available only in SQL Server 2017 CTP 2.0, implements the most popular functions in RevoScaleR, such as remote compute contexts, and many algorithms that support distributed processing.

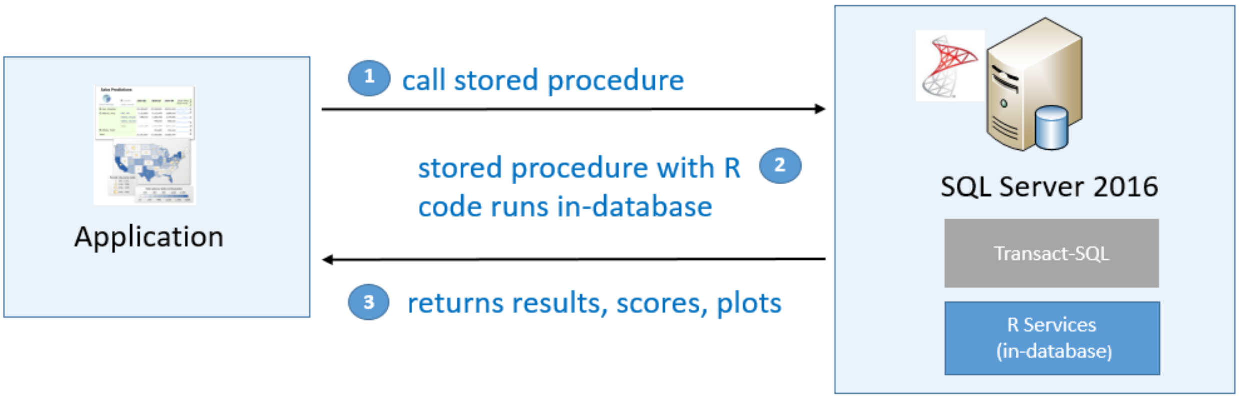
- Choose the best language for the task. R is best for statistical computations that are difficult to implement using SQL. For set-based operations over data, leverage the power of SQL Server to achieve maximum performance. Use the in-memory database engine for very fast computations over columns.

Resources

- Performance Case Study
- R and Data Optimization

Step 4. Deploy and Consume

After the R 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 sessions. You also have control over the users who can execute R scripts, and you can specify which databases can be accessed by R code. You can control the amount of resources allocated to the R 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 securty features in SQL Server, the deployment process might include these tasks:

- Converting your R 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

Resources

- Resource Governance for R
- R Package Management for SQL Server

Quick Starts

Read this walkthrough to understand the full workflow, from exploring data to creating a model and deploying it to SQL Server.

- Data Science End-to-End Walkthrough

Learn how to use the RevoScaleR package for scalable and high performance analysis.

- Data Science Deep Dive: Using the RevoScaleR Packages

Especially for the data developer -- all R code provided! Learn how to embed R in SQL stored procedures to create or train models, and get predictions from a stored model.

- In-Database Advanced Analytics for SQL Developers

Learn the syntax for calling R from a stored procedure.

- Using R Code in Transact-SQL

Solutions

For more samples, including industry=specific solution templates, see SQL Server Machine Learning Tutorials.

Set up SQL Server R Services (In-Database)

4/29/2017 • 11 min to read • Edit Online

This topic describes how to set up machine learning services for the R language, in SQL Server 2017 or later, by running the SQL Server setup wizard.

SQL Server setup provides these options for installing R:

- Install R in the SQL Server database

  This option lets you run R scripts inside SQL Server, or lets the server act as a remote compute context for R scripts run from an external connection.

- Install a standalone server

  This option creates a development environment for machine learning solutions that runs separately from SQL Server. We recommend installing this on a different server than the one hosting SQL Server. For more information on this option, see Create a Standalone R Server.

Installation process

1. Install the external scripting feature
2. Enable the feature and restart the computer
3. Verify that external script execution works
4. Additional configuration of accounts, packages, or databases

Prerequisites

- Avoid installing both R Server and R Services in a single installation. Installing R Server (Standalone) is typically done because you want to create an environment that a data scientist or developer can use to connect to SQL Server and deploy R solutions. Therefore, there is no need to install both on the same computer.

- If you have installed 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 should uninstall them first. Side-by-side install is not supported. For help uninstalling previous versions, see Upgrade and Installation FAQ for SQL Server R Services

- You cannot install machine learning serviceson a failover cluster. The reason is that the security mechanism used for isolating external script processes is not compatible with a Windows Server failover cluster environment. As a workaround, you can use replication to copy necessary tables to a standalone SQL Server instance with R Services, or install R Services (In-Database) on a standalone computer that uses Always On and is part of an availability group. .

> **IMPORTANT**
>
> After setup is complete, some additional steps are required to enable machine learning feature. You might also need to give users permissions to specific databases, change or configure accounts, or set up a remote data science client.

Step 1: Install the extensibility features and choose a machine learning language

To add machine learning requires that you install SQL Server 2016 or later.

1. Run SQL Server setup.

   For information about how to do unattended installs, see Unattended Installs of SQL Server R Services.

2. On the **Installation** tab, click **New SQL Server stand-alone installation or add features to an existing installation**.

3. On the **Feature Selection** page, select the following options, to install the database services used by R jobs and installs the extensions that support external scripts and processes:

   SQL Server 2016

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

   SQL Server 2017

   - Select **Database Engine Services**
   - Select **Machine Learning Services (In-Database)**
   - Select at least one machine learning language to enable. You can select just R, or add both R and Python.

   At least one instance of the database engine is required to use R Services (In-Database). You can use either a default or named instance.

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

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

   > **NOTE**
   >
   > If the computer you are using does not have Internet access, you can pause setup at this point to download the installers separately as described here: Installing R Components without Internet Access

   Click **Accept**, wait until the **Next** button becomes active, and then click **Next**.

5. On the **Ready to Install** page, verify that these selections are included, and click **Install**.

   SQL Server 2016

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

   SQL Server 2017

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

6. When installation is complete, restart the computer as instructed.

Step 2: Enable External Script Services

1. Open SQL Server Management Studio. If it is not already installed, you can rerun the SQL Server setup wizard to open a download link and install it.

2. Connect to the instance where you installed machine learning, 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, to reduce the surface area, and must be explicitly enabled by an administrator.

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

```
Exec sp_configure  'external scripts enabled', 1
Reconfigure  with override
```

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

   You can restart the service using the **Services** panel in Control Panel, or by using SQL Server Configuration Manager.

Step 3. Verify that Script Execution Works Locally

Verify that the external script execution service is enabled.

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 and verify that the Launchpad service for your instance is running. If you install multiple instances, each instance has its own Launchpad service.

3. Open a new Query window in SQL Server Management Studio, and run a simple R script, like 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
```

   Results

   hello
   1

   If the command executes without an error, go on to the next steps. If you get an error, see this article for a list of some common problems: Upgrade and Installation FAQ.

Next Steps

After you have verified that the script execution feature works in SQL Server, you can run R commands from SQL Server Management Studio, Visual Studio Code, or any other client that can send T-SQL statements to the server.

However, if you intend to run commands from a remote R development client, you might need to make additional changes. For more information, see Additional Configuration.

Tutorials

To get started with some simple examples, and learn the basics of how R works with SQL Server, see Using R Code in Transact-SQL.

To view advanced machine learning walkthroughs based on real-world scenarios, see R Tutorials

Install R packages

Take a minute to install any additional R packages you'll be using.

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 compouter, or if you installed packages to user libraries, you won't be able to use those packages from T-SQL. For more information, see Install Additional R Packages on SQL Server.

YOu can also set up user groups to share packages on a per-database level, or configura database roles to enable users to install their own packages. For more information, see Package Management.

Additional Configuration

Depending on your use case for R in SQL Server, you might need to make additional changes to the server, the firewall, the accounts used by the service, or to dtabase permissions.

For example, you might need to access SQL Server data from a remote R development terminal, perform ODBC calls from your R code, or convert R code to stored procedures. If you have additional security restrictions, you will need to ensure that the Launchpad services accounts can access the database. If you will be running R scripts from a remote computer, you will need to give the remote users database access and permission to run external scripts.

> **NOTE**
>
> Not all of the listed changes might be required. However, please review all items to see which might be applicable to your scenario.

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 an R script from an external client, SQL Server will activate an available worker account, map it to the identity of the calling user, and run the R script on behalf of the user. This is a new service of the database engine that supports 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 typucally 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, these worker accounts must be given permission to log into 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, click **Search**.
3. Click **Object Types** and select **Groups**. Deselect everything else.
4. In Enter the object name to select, type *SQLRUserGroup* and click **Check Names**.
5. The name of the local group associated with the instance's Launchpad service should resolve to something like *instancename\SQLRUserGroup*. Click **OK**.
6. By default, the login is assigned to the **public** role and has permission to connect to the database engine.
7. Click **OK**.

NOTE

If you use a SQL login for running R 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 are running R scripts in your own instance, you are typically executing scripts as an administrator and thus have implicit permission over various operations and all data in the database, as well as the ability to install new R packages as needed.

However, in an enterprise scenario, most users, including those accessing the database using SQL logins, do not have such elevated permissions. Therefore, for each user that will be running external scripts, you must grant the user permissions to run R scripts in each database where R 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 the installation status of R Services. For more information, see Monitor R Services using Custom Reports.

Ensure that the SQL Server supports remote connections

If you cannot connect from a remote computer, check whether the server itself permits remote connections. Sometimes remote connections are disabled by default.

Check whether the firewall allows access to SQL Server. By default, the port used by SQL Server is often blocked by the firewall. If you are using the Windows firewall, see [Configure Windows Firewall for Database Engine Access](https://docs.microsoft.com/sql/database-engine/configure-windows/configure-a-windows-firewall-for-database-engine-access

Add more worker accounts

If you think you will 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 R Services.

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

While running R scripts, the user account or SQL login might need to read data from other databases, create new tables to store results, and write data into tables.

For each user account or SQL login that will be executing R scripts, be sure that the account or login has **db_datareader**, **db_datawriter**, or **db_ddladmin** permissions on the specific database.

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 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 this by adding *Connect to* and *SELECT* permissions, or by adding the login to the **db_datareader** role. If you need to create objects, you will need **DDL_admin** rights. To save data to tables, add the login to the **db_datawriter** role.

- For Windows authentication: You must configure an ODBC data source on the data science client that specifies the instance name and other connection information. For more information, see Using the ODBC Data Source Administrator.

Optimize the server for R

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

To ensure that R tasks are prioritized and resourced appropriately, we recommend that you use Resource Governor to configure an external resource pool specific for R operation. You might also wish to change the amount of memory allocated to the SQL Server database engine, or increase the number of accounts running under the SQL Server Trusted Launchpad service.

- Configure a resource pool for managing external resources

  CREATE EXTERNAL RESOURCE POOL (Transact-SQL)

- Change the amount of memory reserved for the database engine

  Server Memory Server Configuration Options

- Change the number of R accounts that can be started by SQL Server Trusted Launchpad

  Modify the User Account Pool for SQL Server R Services

If you are using Standard Edition and do not have Resource Governor, you can use DMVs and Extended Events, as well as Windows event monitoring, to help you manage server resources used by R. For more information, see Monitoring and Managing R Services.

More tips and known issues

Run into trouble? Trying to upgrade from a pre-release version of R Services (In-Database)? See this list of common issues.

- Upgrade and Installation FAQ (SQL Server R Services)

  Try these custom reports, to check the installation status of the instance and fix common issues.

- Custom Reports for SQL Server R Services

See Also

Unattended Installs of R Machine Learning Services (In-Database)

4/29/2017 • 3 min to read • Edit Online

This topic describes how to use command-line arguments with SQL Server setup to install R together with an instance of the database engine, in quiet mode. In an unattended installation, you do not use the interactive features of the setup wizard, and must provide all arguments required to complete installation, including licensing agreements for SQL Server and for machine learning components.

> **IMPORTANT**
>
> The setup process is different in SQL Server 2016 and SQL Server 2017.
>
> - SQL Server 2016 supports R integration using the feature, **SQL Server R Services**. See this section for setup arguments.
> - SQL Server 2017 includes support for both R and Python, using the feature **Machine Learning Services (In-Database). See this section for R Service setup arguments.
> - The prerequisites are the same. Be sure to install the R components beforehand if the computer does not have Internet access.
> - In both SQL Server 2016 and SQL Server 2017, additional steps are required after setup to enable the feature.

Prerequisites

- You must install the database engine on each instance where you will use R.
- If you are performing an offline install, you must download the required R components in advance, and copy them to a local folder. For download locations, see Installing R Components without Internet Access.
- There is a new parameter, */IACCEPTROPENLICENSETERMS*, that indicates you have accepted the license terms for using the open source R components. If you do not include this parameter in your command line, setup will fail.
- To complete setup without having to respond to prompts, make sure that you have identified all required arguments, including those for R and SQL Server licensing, and for any other features that you might want to install.

> **NOTE**
>
> The Mixed security mode that supports SQL logins was required in early releases. However, you might consider enabling Mixed Mode authentication to support solution development by data scientists using a SQL login.

Unattended Install of R Machine Learning Services in SQL Server 2017

The following example shows the **minimum** required features to specify in the command line when performing a silent, unattended install of Machine Services (In-Database). Requires SQL Server 2017.

1. Open an elevated command prompt, and run the following command:

```
Setup.exe /q /ACTION=Install /FEATURES=SQL,AdvancedAnalytics, SQL_INST_MR /INSTANCENAME=MSSQLSERVER
/SECURITYMODE=SQL /SAPWD="%password%" /SQLSYSADMINACCOUNTS="<username>" /IACCEPTSQLSERVERLICENSETERMS
/IACCEPTROPENLICENSETERMS
```

Note the flags required for R in SQL Server 2017: `AdvancedAnalytics`, `SQL_INST_MR`, and `IACCEPTRLICENSETERMS`.

2. Restart the server.
3. Perform the post-installation configuration steps as described in this section. Another restart of the SQL Server services will be required.

Unattended Install of R Services (In-Database) in SQL Server 2016

The following example shows the **minimum** required features to specify in the command line when performing a silent, unattended install of SQL Server R Services. Requires SQL Server 2016.

1. Open an elevated command prompt, and run the following command:

```
Setup.exe /q /ACTION=Install /FEATURES=SQL,AdvancedAnalytics /INSTANCENAME=MSSQLSERVER /SECURITYMODE=SQL
/SAPWD="%password%" /SQLSYSADMINACCOUNTS="<username>" /IACCEPTSQLSERVERLICENSETERMS
/IACCEPTROPENLICENSETERMS
```

Note the flags required for 2016 R SERVICES: `AdvancedAnalytics` and `IACCEPTRLICENSETERMS`.

2. Restart the server.
3. Perform the post-installation configuration steps as described in this section. Another restart of the SQL Server services will be required.

Post-Installation Steps

1. After installation is complete, open a new Query window in SQL Server Management Studio, and run the following command to enable the feature.

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

NOTE

You must explicitly enable the feature and reconfigure; otherwise, it will not be possible to invoke external scripts even if the feature has been installed.

2. Restart the SQL Server service for the reconfigured instance. Doing so will automatically restart the related SQL Server Trusted Launchpad service as well.

3. Additional steps might be required if you have a custom security configuration, or if you will be using the SQL Server to support remote compute contexts. For more information, see Upgrade and Installation FAQ.

Architecture Overview (SQL Server R Services)

4/29/2017 • 2 min to read • Edit Online

This section provides an overview of the architecture of SQL Server 2016 R Services, and 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 topic 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 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 or Python 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. 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

New 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, pass data between R and the database engine, and 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 rremote or local users, and monitor and manage the resources consumed.

For details, see Security for R

See Also

Machine Learning Tutorials

R Interoperability in SQL Server R Services

4/19/2017 • 3 min to read • Edit Online

This topic focuses on the mechanism for running for R within R Services (In-Database), and describes the differences between Microsoft R and open source R. 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 `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library`.

Similarly, the R tools associated with the default instance would be located in `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\bin`,

For more information about the base distribution, see Packages installed with Microsoft R Open

Additional R Packages

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, and libraries that support 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. An example of how to use the XDF format is provided in this tutorial: Data Science Deep Dive

- **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.

See Also

Architecture Overview

New Components in SQL Server to Support R Services

Security Overview

Components in SQL Server to Support R

4/29/2017 • 6 min to read • Edit Online

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

New 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 ScaleR 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");
  ```

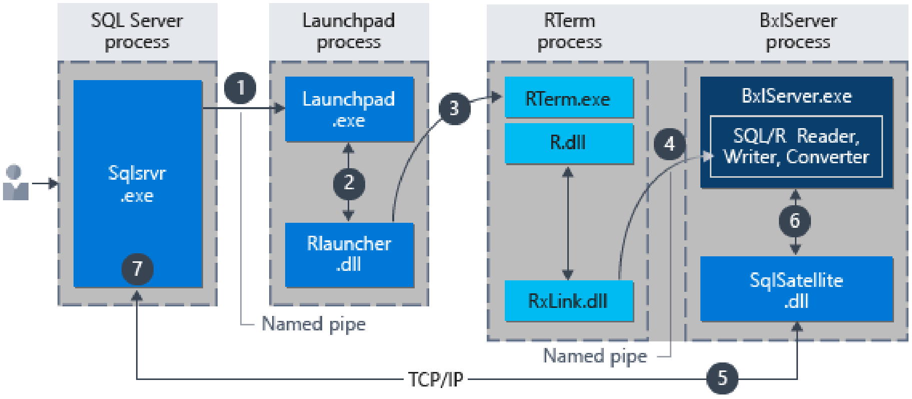
- **Other protocols** Processes that might need to work in "chunks" or transfer data back to a remote client can also use the .XDF format supported by Microsoft R. Actual data transfer is via encoded blobs.

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)

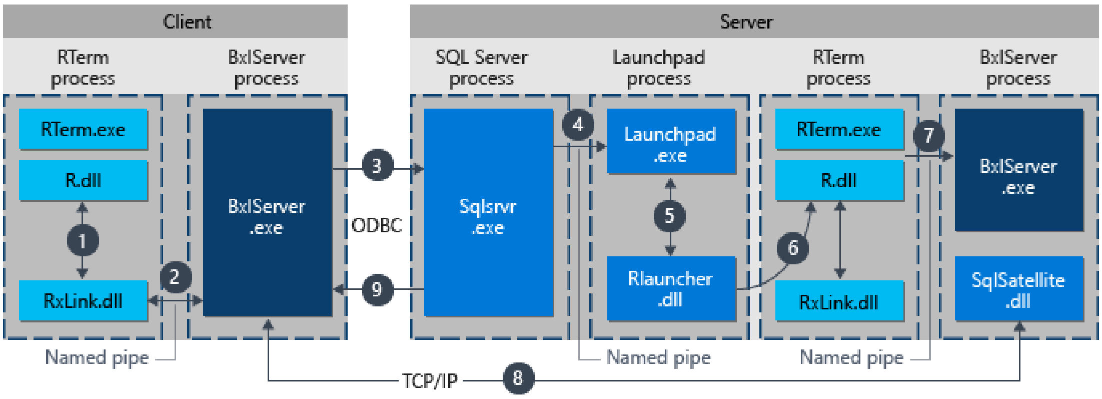
R code that is run from "inside" SQL Server is executed by calling a stored procedure. Thus, any application that can make a stored procedure call can initiate execution of R code. Thereafter SQL Server manages the execution of R code as summarized in the following diagram.



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 ScaleR 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 is 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.

See Also

Architecture Overview

Security Overview (SQL Server R Services)

4/29/2017 • 4 min to read • Edit Online

This topic 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 two 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.

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 New Components in SQL Server to Support R Integration.

> **NOTE**
>
> For Launchpad to manage the worker accounts and execute R jobs, the group that contains the worker accounts, SQLRUserGroup, must have "Allow Log on locally" permissions; otherwise R Services might not work. By default, this right is given to all new local users, but in some organizations stricter group policies might be enforced, which prevent the worker accounts from connecting to SQL Server to perform R jobs.

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 R Services.

Security Isolation for Multiple External Scripts

The isolation mechanism is based on 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

Monitoring R Services

4/19/2017 • 1 min to read • Edit Online

This section provides information about these common monitoring tasks when using R Services (In-Database):

- Identifying server resources being used by R
- Monitoring the status of R tools
- Determining the jobs that are executing using the R runtime
- Identifying remote R jobs that are using the SQL Server instance as the compute context
- Correlating long-running queries with R jobs
- Terminating R processes

  The topic also describes the tools installed on the SQL Server computer as part of R setup, and lists the DMVs and Extended Events related to R Services (In-Database).

In This Section

DMVs for SQL Server R Services

Extended Events for SQL Server R Services

Using R Code Profiling Functions

Monitor R Services Using Custom Reports in Management Studio

See Also

Resource Governance for R Services

Machine Learning Services with Python

4/29/2017 • 1 min to read • Edit Online

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 natural language processing. Now, this popular language is supported in SQL Server 2017 CTP 2.0.

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, and 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 new **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 setup:

- **Machine Learning Services (In-Database):** Install this feature, together with the SQL Server database engine, to enable secure execution of R 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 support and use the deploy and consume features of mrsdeploy.

  Do not install this feature on the same computer that is running SQL Server Machine Learning Services.

Additional Resources

Set Up Python Machine Learning Services In-Database

Tutorials

Set up Python Machine Learning Services (In-Database)

4/25/2017 • 11 min to read • Edit Online

You install the components required for using Python by running the SQL Server setup wizard and following the interactive prompts as described in this topic.

Overview of the setup process

- Be sure to install the database engine. An instance of SQL Server is required to run Python scripts in-database.
- Choose the **Machine Learning Services** feature, and select **Python** as the language.
- After installation is complete, reconfigure the instance to allow execution of scripts that use an external executable.
- You might need to make some additional configuration of the worker account pool that is used to run Python.
- Each configuration change requires a restart of the Trusted Launchpad service.

To perform an unattended installation, use the command-line options for SQL Server setup and the arguments specific to Python, as described here: Unattended Installs of SQL Server with Python Machine Learning Services.

Prerequisites

- SQL Server 2017 is required. Python integration is not supported on previous versions of SQL Server.
- Additional prerequisites, such as .NET Core, are installed as part of the Python component setup.
- The **Shared Features** section contains a separate installation option, **Machine Learning Server (Standalone)**. We recommend that you **do not** install this on the same computer as a SQL Server instance that uses R Services or Python Services. Instead, install the Machine Learning Server (Standalone) on a separate computer that you will use for developing and testing your R solutions.
- You cannot install Machine Learning with Python Services on a failover cluster. The reason is that the security mechanism used for isolating Python processes is not compatible with a Windows Server failover cluster environment.

  As a workaround, you can use replication to copy necessary tables to a standalone SQL Server instance that uses Python Services, or you can install Machine Learning with Python Services on a standalone computer that uses Always On and is part of an availability group.
- Side-by-side installation with other versions of Python is possible, because the SQL Server instance uses its own copy of the Anaconda distribution. However, running code using Python on the SQL Server computer outside of SQL Server can lead to various problems:
  - You use a different library and different executable and get different results than when running in SQL Server.
  - 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 additional post-configuration steps described in this topic. These include enabling SQL Server to use external scripts, and adding accounts required for SQL Server to run Python jobs on your behalf.

Step 1: Install Machine Learning Services (In-Database) on SQL Server

1. Run the setup wizard for SQL Server 2017.

2. On the **Installation** tab, click **New SQL Server stand-alone installation or add features to an existing installation**.

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

   - **Database Engine Services**

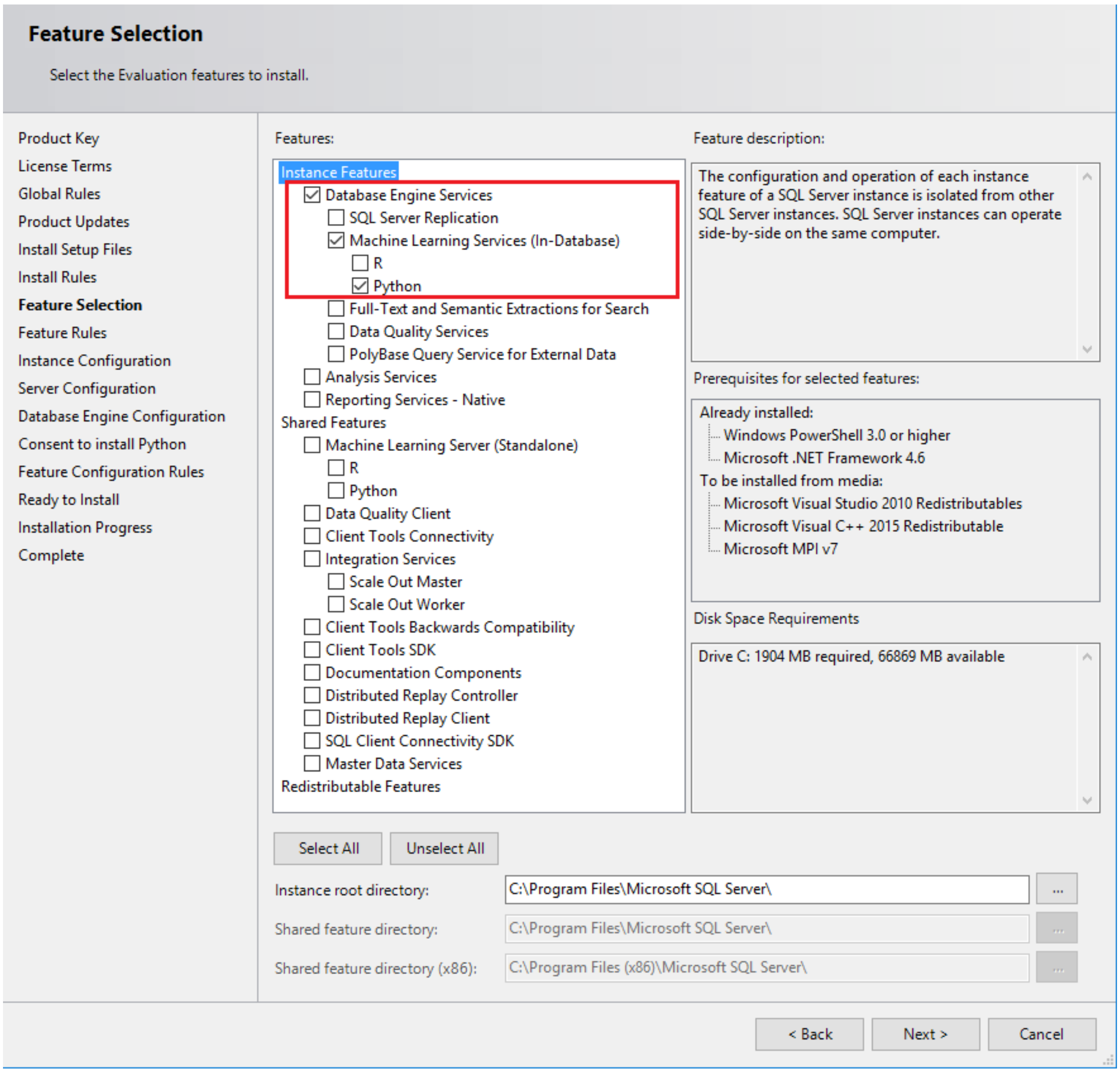
     To use Python with SQL Server, you must install an instance of the database engine. You can use either a default or named instance.

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

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

   - **Python** Check this option to get the Python 3.5 executable and select libraries from the Anaconda distribution. We recommend you install only one language per instance.

   > **NOTE**
   >
   > Do not select the option in **Shared Features** for **Microsoft R Server (Standalone)**. Use this option in a separate installation if you need to add the Machine Learning components to a different computer that is used for R development, such as your data scientist's laptop.



Feature Selection

Select the Evaluation features to install.

Features:

Instance Features
- ☑ Database Engine Services
  - □ SQL Server Replication
  - ☑ Machine Learning Services (In-Database)
    - □ R
    - ☑ Python
  - □ Full-Text and Semantic Extractions for Search
  - □ Data Quality Services
  - □ PolyBase Query Service for External Data
- □ Analysis Services
- □ Reporting Services - Native

Shared Features
- □ Machine Learning Server (Standalone)
  - □ R
  - □ Python
- □ Data Quality Client
- □ Client Tools Connectivity
- □ Integration Services
  - □ Scale Out Master
  - □ Scale Out Worker
- □ Client Tools Backwards Compatibility
- □ Client Tools SDK
- □ Documentation Components
- □ Distributed Replay Controller
- □ Distributed Replay Client
- □ SQL Client Connectivity SDK
- □ Master Data Services

Redistributable Features

Feature description:

The configuration and operation of each instance feature of a SQL Server instance is isolated from other SQL Server instances. SQL Server instances can operate side-by-side on the same computer.

Prerequisites for selected features:

Already installed:
- Windows PowerShell 3.0 or higher
- Microsoft .NET Framework 4.6

To be installed from media:
- Microsoft Visual Studio 2010 Redistributables
- Microsoft Visual C++ 2015 Redistributable
- Microsoft MPI v7

Disk Space Requirements

Drive C: 1904 MB required, 66869 MB available

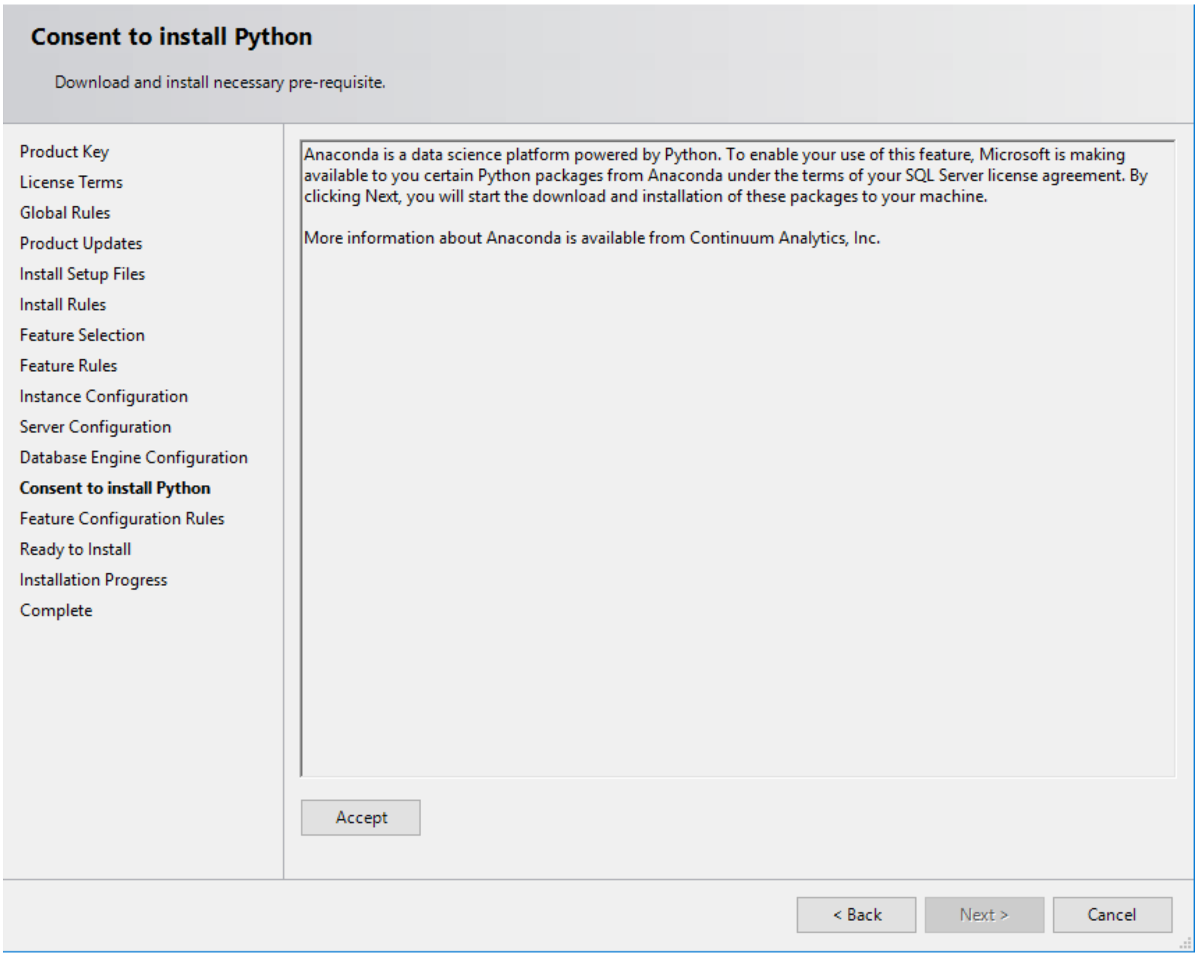
Instance root directory: C:\Program Files\Microsoft SQL Server\

Shared feature directory: C:\Program Files\Microsoft SQL Server\

Shared feature directory (x86): C:\Program Files (x86)\Microsoft SQL Server\

4. On the page, **Consent to Install Python**, click **Accept**.

This license agreement is required to download the Python executable, Python packages from Anaconda.



Consent to install Python

Download and install necessary pre-requisite.

Anaconda is a data science platform powered by Python. To enable your use of this feature, Microsoft is making available to you certain Python packages from Anaconda under the terms of your SQL Server license agreement. By clicking Next, you will start the download and installation of these packages to your machine.

More information about Anaconda is available from Continuum Analytics, Inc.

NOTE

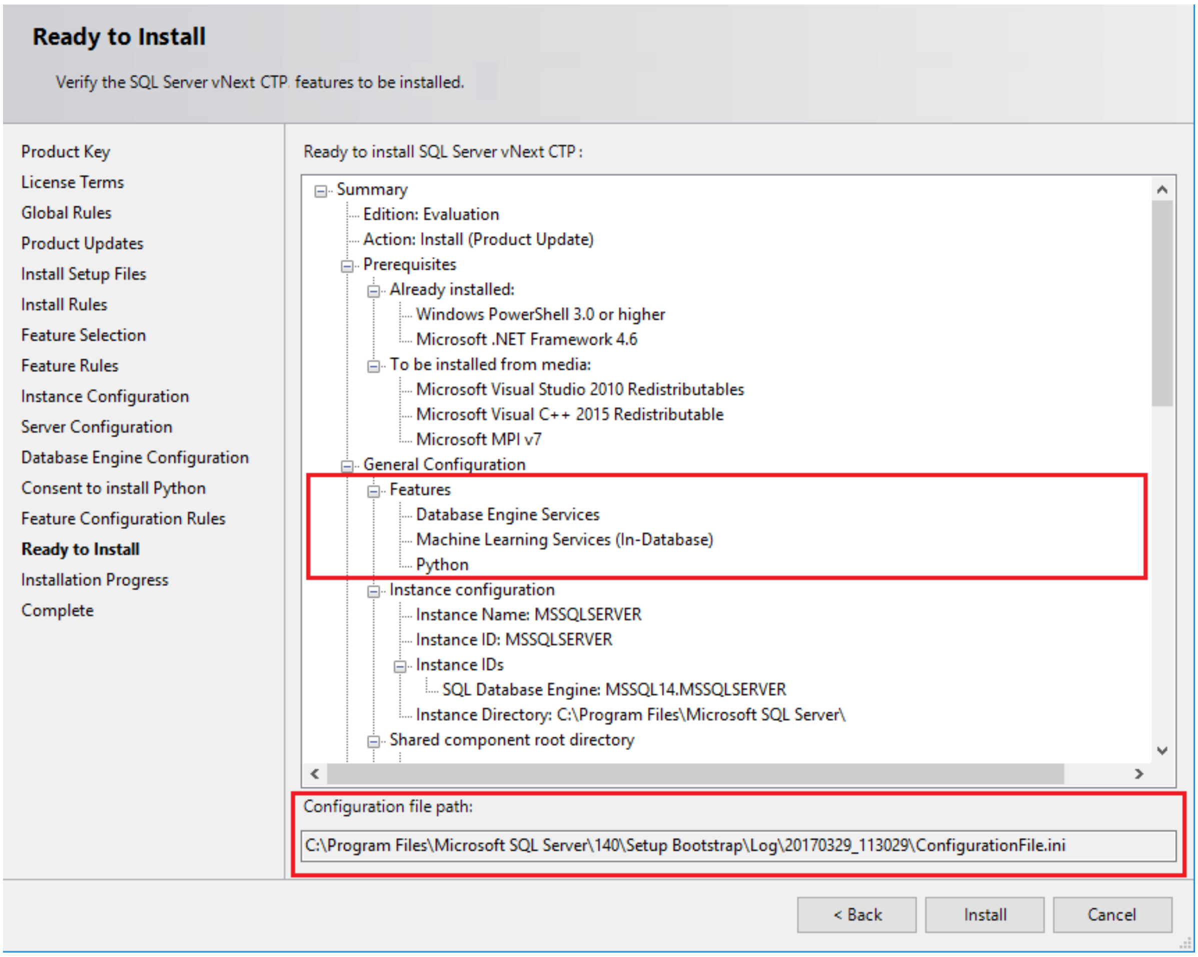
If the computer you are using does not have Internet access, you can pause setup at this point to download the installers separately as described here: Installing Components without Internet Access

Click **Accept**, wait until the **Next** button becomes active, and then click **Next**.

5. On the **Ready to Install** page, verify that these selections are included, and click **Install**.

- Database Engine Services
- Machine Learning Services (In-Database)
- Python

These selections represent the minimum configuration required to use Python with SQL Server.



Ready to install SQL Server vNext CTP :

- Summary
  - Edition: Evaluation
  - Action: Install (Product Update)
  - Prerequisites
    - Already installed:
      - Windows PowerShell 3.0 or higher
      - Microsoft .NET Framework 4.6
    - To be installed from media:
      - Microsoft Visual Studio 2010 Redistributables
      - Microsoft Visual C++ 2015 Redistributable
      - Microsoft MPI v7
  - General Configuration
    - Features
      - Database Engine Services
      - Machine Learning Services (In-Database)
      - Python
    - Instance configuration
      - Instance Name: MSSQLSERVER
      - Instance ID: MSSQLSERVER
      - Instance IDs
        - SQL Database Engine: MSSQL14.MSSQLSERVER
      - Instance Directory: C:\Program Files\Microsoft SQL Server\
    - Shared component root directory

Configuration file path:

C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log\20170329_113029\ConfigurationFile.ini

Optionally, make a 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.

6. When installation is complete, restart the computer.

Step 2: Enable Python script execution

1. Open SQL Server Management Studio. If it is not already installed, you can re-run the SQL Server setup wizard to open a download link and install it.

2. Connect to the instance where you installed Machine Learning Services, 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, to reduce the surface area. The feature must be explicitly enabled by an administrator before you can run R or Python scripts.

3. To enable the external scripting feature that supports Python, run the following statement.

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

Note that this is exactly the same process that is used to enable R, because the underlying extensibility feature supports both languages.

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

You can restart the service using the **Services** panel in Control Panel, or by using [SQL Server Configuration Manager](https://docs.microsoft.com/en-us/relational-databases/sql-server-configuration-manager.

Step 3. Verify that the external script execution feature is running

Take a moment to verify that all components used to launch Python 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 that the Launchpad service for your instance is running. If the Launchpad is not running, restart the service.

   If you have installed multiple instances of SQL Server, using either R or Python, each instance has its own Launchpad service.

   However, if you install R and Python on a single instance, only one Launchpad is installed, but a separate language-specific launcher DLL is added for each language. For more information, see Components to Support Python Integration

3. If Launchpad is running, you should be able to run simple Python scripts like the following in SQL Server Management Studio:

   ```
   exec sp_execute_external_script  @language =N'Python',
   @script=N'OutputDataSet=InputDataSet',
   @input_data_1 = N'SELECT 1 as col'
   ```

   Results

   col 1

Troubleshooting and Additional Steps

If this command 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 not, review the following list and see if changes to the configuration of the service or database might be needed.

It is important to note that not all of the listed changes are required, and none might be required. It depends on your security schema, where you installed SQL Server, and how you expect users to connect to the database and run external scripts.

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 will activate an available worker account, map it to the identity of the calling user, and run the script on behalf of the user.

This is called called *implied authentication*, and is a service of the database engine that 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 typically more than enough for running external script jobs.

> IMPORTANT
>
> The worker group is named SQLRUserGroup regardless of the type of script you are running. There is a single group for each instance.

If you need to run R scripts from a remote data science client and are using Windows authentication, these worker accounts must be given permission to log into 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, click **Search**.
3. Click **Object Types** and select **Groups**. Deselect everything else.
4. In Enter the object name to select, type *SQLRUserGroup* and click **Check Names**.
5. The name of the local group associated with the instance's Launchpad service should resolve to something like *instancename\SQLRUserGroup*. Click **OK**.
6. By default, the login is assigned to the **public** role and has permission to connect to the database engine.
7. Click **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 are running R scripts in your own instance, you are typically executing scripts as an administrator and thus have implicit permission over various operations and all data in the database, as well as the ability to install new R packages as needed.

However, in an enterprise scenario, most users, including those accessing the database using SQL logins, do not have such elevated permissions. Therefore, for each user that will be running external scripts, you must grant users of machine learning services the permission to run scripts in each database where R or Python will be used.

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

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

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 R Services.

Give your users read, write, or DDL permissions to databases

While running scripts, the user account or SQL login might need to read data from other databases, create new tables to store results, and write data into tables.

For each user account or SQL login that will be executing R or Python scripts, be sure that the account or login has **db_datareader**, **db_datawriter**, or **db_ddladmin** permissions on the specific database.

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.

Ensure that the SQL Server supports remote connections

If you cannot connect from a remote computer, check whether the firewall allows access to SQL Server. In a default installation, remote connections might be disabled, or the specific port used by SQL Server might be blocked by the firewall. For more information, see Configure Windows Firewall for Database Engine Access

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

If you create a machine learning solution on a data science client computer and need to run code 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 this by adding *Connect to* and *SELECT* permissions, or by adding the login to the **db_datareader** role. If you need to create objects, you will need **DDL_admin** rights. To save data to tables, add the login to the **db_datawriter** role.

- For Windows authentication: You must configure an ODBC data source on the data science client that specifies the instance name and other connection information. For more information, see Using the ODBC Data Source Administrator.

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 supported by the database engine, which might include ETL processes, reporting, auditing, and applications that use SQL Server data.

As a result, if you use the default settings created by SQ Server setup, you might find that resources for running external scripts are restricted or throttled, particularly in memory-intensive operations.

If machine learning is a priority, you need to change default database settings to ensure that external script jobs are prioritized and resourced appropriately. These changes might include:

- Reduce the amount of memory allocated to the SQL Server database engine
- Increase the number of accounts running under the SQL Server Trusted Launchpad service. This does not increase the number of resources, but does increase the number of scripts that can run concurrently.

If you have Enterprise Edition, we recommend that you use Resource Governor to configure an external resource pool specific for Python. See these articles for more information:

- Configure a resource pool for managing external resources

  CREATE EXTERNAL RESOURCE POOL (Transact-SQL)

- Change the amount of memory reserved for the database engine

  Server Memory Server Configuration Options

- Change the number of worker accounts that can be started by SQL Server Trusted Launchpad

  Modify the User Account Pool for SQL Server R Services

If you are using Standard Edition and do not have Resource Governor, you can use DMVs and extended events, as well as Windows event monitoring, to help you manage server resources. For more information, see Monitoring and Managing R Services.

See Also

Using Python in T-SQL

Create a Python Model using revoscapepy

Architecture Overview - Machine Learning Services with Python

4/29/2017 • 1 min to read • Edit Online

This topic 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 topics.

> **IMPORTANT**
>
> Support for Python is available beginning with SQL Server 2017. This is a pre-release feature and 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 ensure near-complete compatibility with standard Python solutions. Python runs in a separate process from SQL Server, to ensure that database operations are not compromised.

For more information about the interaction of SQL Server with Python, see Python Interoperability

New 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.

See Also

Run Python using T-SQl

Python Interoperability

4/25/2017 • 2 min to read • Edit Online

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

> **NOTE**
>
> Support for Python is a pre-release feature and is still under development.

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
```

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 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.

Because support for Python is a pre-release feature and still under development, the **revoscalepy** library currently includes only a subset of the RevoScaleR functionality.

Future additions might include the Microsoft Cognitive Toolkit. Formerly known as CNTK, this library supports a variety of neural network models, including convolutional networks (CNN), recurrent networks (RNN), and Long Short Term Memory networks (LSTM).

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

New components in SQL Server to Support Python Integration

4/29/2017 • 5 min to read • Edit Online

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 Set up Machine LEarnign Services

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 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.

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

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

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 will map the identity of the user issuing the remote 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.

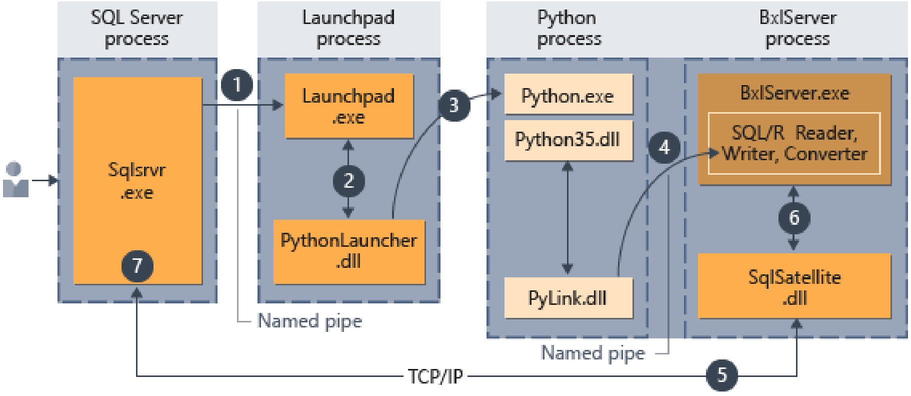
Interaction of Components

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.



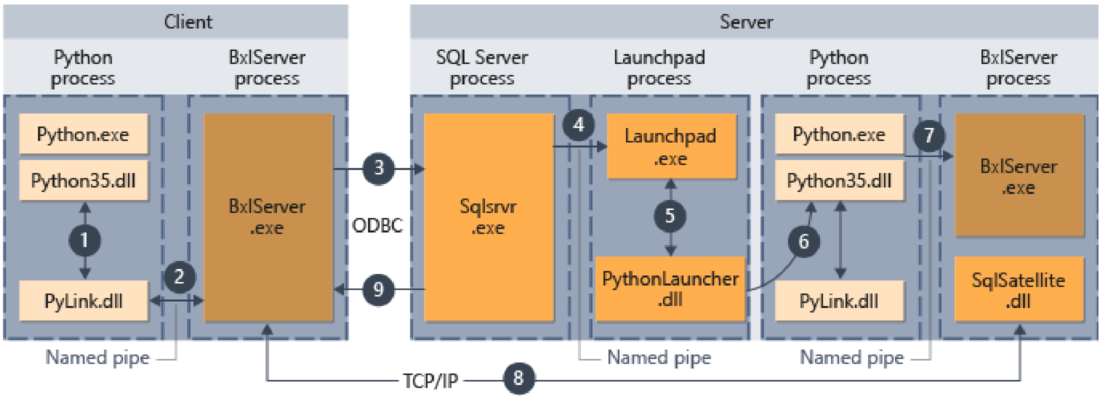
1. A request for the R runtime is indicated by the parameter *@language='Python'* passed to the stored procedure. SQL Server sends this request to the Launchpad service.
2. The Launchpad service starts the appropriate launcher; in this case, PythonLauncher.
3. PythonLauncher starts the external Python35 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 following diagram summarizes the overall workflow when scripts are sent from a remote computer.



1. For function that are supported in RevoScalePy functions, 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 is 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.

Security Overview

4/19/2017 • 4 min to read • Edit Online

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. The login or user account identifies the *security principal*, who must have permission to access the database where data is retrieved from. Depending on whether the Python script creates new objects or writes new data, the user might need permissions to create tables, write data, or create custom functions or stored procedures.

Therefore, it is a strict requirement that each person who runs Python code in SQL Server must be mapped to a login or account in the database. This restriction applies regardless of whether the 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 will get your credentials and map it 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 New Components in SQL Server to Support Python Integration.

NOTE

For Launchpad to manage the worker accounts and execute Python jobs, the group that contains the worker accounts, *SQLRUserGroup*, must have "Allow Log on locally" permissions; otherwise the Python run-time might not be started. By default, this right is given to all new local users, but in some organizations stricter group policies might be enforced, which prevent the worker accounts from connecting to SQL Server to Python jobs.

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 R Services.

Security Isolation for Multiple External Scripts

The isolation mechanism is based on 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

Monitoring

4/29/2017 • 1 min to read • Edit Online

NOTE

Python support is a new feature in SQL Server 2017 and is in prerelease.

Support for Python is a new feature in SQL Server, available in SQL Server 2017 CTP 2.0.

However, the overall framework for external script execution and data security is the same as for R. You can monitor any external script, including one that runs Python, using the same framework that was provided for R script execution in SQL Server 2016. For more information, see Monitoring R Solutions.

See these related topics for more information:

- Managing and Monitoring R solutions
- Resource Governance for R

R Server (Standalone)

4/29/2017 • 3 min to read • Edit Online

In SQL Server 2016, Microsoft released **R Server (Standalone)**, as part of its platform for supporting enterprise-class analytics. Microsoft R Server provided scalability and security to the R language, and addressed the in-memory limitations of open source R. Like SQL Server R Services, Microsoft R Server (Standalone) provides parallel and chunked processing of data, enabling R users to use data much bigger than can fit in memory.

In SQL Server 2017, support has been added for the Python language, which is another popular choice for advanced machine learning and text processing tasks. To reflect support for this broader set of languages, we've also renamed it to **Microsoft Machine Learning Server (Standalone)**.

What's Included in Microsoft R Server

You can use Microsoft R Server for distributed computing on multiple platforms. When you install from SQL Server setup, you get the Windows-based server and all the required tools for publishing and deploying models. For more information about other platforms, see these resources in the MSDN library:

- Introducing Microsoft R Server
- R Server for Windows

You can also install Microsoft R Server to use as a development client, to get the RevoScaleR libraries and tools needed to create R solutions that can be deployed to SQL Server.

What's New in Microsoft Machine Learning Server

If you install Machine Learning Services (Standalone) using SQL Server 2017 setup, you now have the option to add the Python language. Naturally, the R language is still a supported option, and you can even install both languages if desired.

In SQL Server 2017 CTP 2.0, the server installation also includes the mrsdeploy package and other utilities used for operationalizing models. For more information, see Operationalization with mrsdeploy.

How to Get Microsoft R Server or Machine Learning Server (Standalone)

There are multiple options for installing Microsoft R Server:

Use the SQL Server setup wizard

Create a Standalone R Server

- Run SQL Server 2016 setup to install **Microsoft R Server (Standalone)**. The R language is added by default.
- Run SQL Server 2017 setup to install **Machine Learning Server (Standalone)** and select either R or Python, or both.

> **IMPORTANT**
>
> The option to install the server is in the **Shared Features** section of setup. Do not install any other commponents.
>
> Preferably, do not install the server on a computer where SQL Server R Services or SQL Server Machine Learning Services has been installed.

Use command-line options for SQL Server setup

Install Microsoft R Server from the Command Line

SQL Server setup supports unattended installs via a rich set of command-line arguments.

Use the standalone installer

You can now use a new Windows installer to set up a new instance of Microsoft R Server or Microsoft Machine Learning Server.

Both Microsoft R Server and Microsoft Machine Learning Server require a SQL Server Enterprise agreement. However, after you run the standalone installer, the support policy for an existing intallation is updated, to use the new Modern Lifecycle policy. This support option ensures that updates to machine learning components are applied more frequently than they would be when using the SQL Server service releases.

- Run Microsoft R Server for Windows.
- Upgrade a SQL Server instance

  You can use the standalone installer to upgrade an instance of SQL Server 2016 R Services to use the latest version of R. When you run the installer, the instance you specify will use the Modern Lifecycle support policy instead, and thus get more frequent updates. This update can only be performed on an existing installation of SQL Server 2016. For more information, see Using SqlBindR to Upgrade an Instance of R Services.

Related Machine Learning Products

- **Azure virtual machines with R Server**: The Azure marketplace includes multiple virtual machine images that include R Server. Creating a new Azure virtual machine is the fastest way to set up a server for operationalizing predictive models. Some images come with scaling and sharing features (formerly known as DeployR) already configured, which makes it easier to embed R analytics inside applications and integrate R with backend systems. For more information, see Provision an R Server Virtual Machine

Resources

For samples, tutorials, and more information about Microsoft R, see Microsoft R Products.

See Also

SQL Server R Services

Getting Started with Microsoft R Server (Standalone)

4/19/2017 • 2 min to read • Edit Online

Microsoft R Server (Standalone) helps you bring the popular open source R language into the enterprise, to enable high-performance analytics solutions and integration with other business applications.

Install Microsoft R Server

How you install Microsoft R Server depends on whether you need to use SQL Server data in your applications. If so, you should install using SQL Server setup. If you will not be using SQL Server data, or don't need to run R code in-database, you can use either SQL Server setup, or the new standalone installer.

- Install Microsoft R Server (Standalone) from SQL Server setup. A separate instance of the R binaries is created for R Server, and the instance is licensed via the SQL Server Enterprise Edition support policy. For more information, see Create a Standalone R Server.
- Use the new standalone Windows installer to create a brand new instance of Microsoft R Server that uses the Microsoft Modern Software Lifecycle support policy. For more information, see Run Microsoft R Server for Windows.
- If you have an existing instance of R Server (Standalone) or R Services that you want to upgrade, you must also download and run the Windows-based installer for the update. For more information, see Run Microsoft R Server for Windows.

Install Additional R Tools

We recommend the free Microsoft R Client (download).

You can also use your preferred R development environment to develop solutions for SQL Server R Services or Microsoft R Server. For details, see Setup or Configure R Tools.

Location of R Server Binaries

Depending on the method you use to install Microsoft R Server, the default location is different. Before you start using your favorite development environment, verify where you installed the R libraries:

- Microsoft R Server installed using the new Windows installer

```
C:\Program Files\Microsoft\R Server\R_SERVER
```

- R Server (Standalone) installed via SQL Server setup

```
C:\Program Files\Microsoft SQL Server\130\R_SERVER
```

- R Services (In-Database)

```
C:\Program Files\Microsoft SQL Server\<instance_name>\R_SERVICES
```

Start Using R on Microsoft R Server

After you have set up the server components and configured your R IDE to use the R Server binaries, you can begin developing your solution using the new APIs, such as the RevoScaleR package, MicrosoftML, and olapR.

To get started with R Server, see this guide in the MSDN Library: R Server - Getting Started

- ScaleR: Explore this collection of 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. For more information, see Explore R and ScaleR in 25 Functions

- MicrosoftML: The MicrosoftML package is a set of new machine learning algorithms and transformations developed at Microsoft that are fast and scalable. For more information, see MicrosoftML functions.

See Also

Getting Started with SQL Server R Services

Create a Standalone R Server

4/29/2017 • 7 min to read • Edit Online

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 R solutions on Windows, and then share the solutions across other platforms. You can also use the server option to set up an environment for building solutions for execution on these supported remote compute contexts:

- An instance of SQL Server running R Services
- An instance of R Server using a Hadoop or Spark cluster
- Teradata in-database analytics
- R Server running in Linux

This topic describes the setup steps in SQL Server setup for Microsoft R Server and Microsoft Machine Learning Server.

Which Should I Install?

Microsoft R Server was first offered as a part of SQL Server 2016 and supports the R language. The last version of Microsoft R Server was 9.0.1. In SQL Server 2017, R Server has been renamed **Microsoft Machine Learning Server**, with added support for Python. The latest version of Microsoft Machine Learning Server is 9.1.0.

Both Microsoft R Server and Microsoft Machine Learning Server require Enterprise Edition.

- Install Microsoft R Server (Standalone) using SQL Server setup
- Install Microsoft Machine Learning Server (Standalone) using SQL Server setup

  Setup requires a SQL Server license and upgrades are typically aligned with the SQL Server release cadence. This ensures that your development tools are in synch with the version running in the SQL Server compute context.

- Install R Server for Windows

  With this option, R Server is installed using the Modern Lifecycle support policy. You can also run this installer after setup to upgrade an instance of SQL Server 2016. Currently, you *cannot* install Python support using this option. To get Python, you must install Machine Learning Server using SQL Server 2017 setup.

How to Install Microsoft Machine Learning Server (Standalone)

1. If you have installed a previous version of Microsoft R Server, we recommend that you uninstall it first.

2. Run SQL Server 2017 setup.

3. Click the **Installation** tab, and select **New Machine Learning Server (Standalone) installation**.

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

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

   - Microsoft Machine Learning Server (Standalone)

   - R and Python are both selected by default.

All other options should be ignored.

6. Accept the license terms for downloading and installing the machine learning components. A separate licensing agreement is required for Microsoft R Open and for Python.

   When the **Accept** button becomes unavailable, you can click **Next**.

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

   If the computer does not have Internet access, you should download the component installers in advance. For more information, see Installing ML components without Internet Access.

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

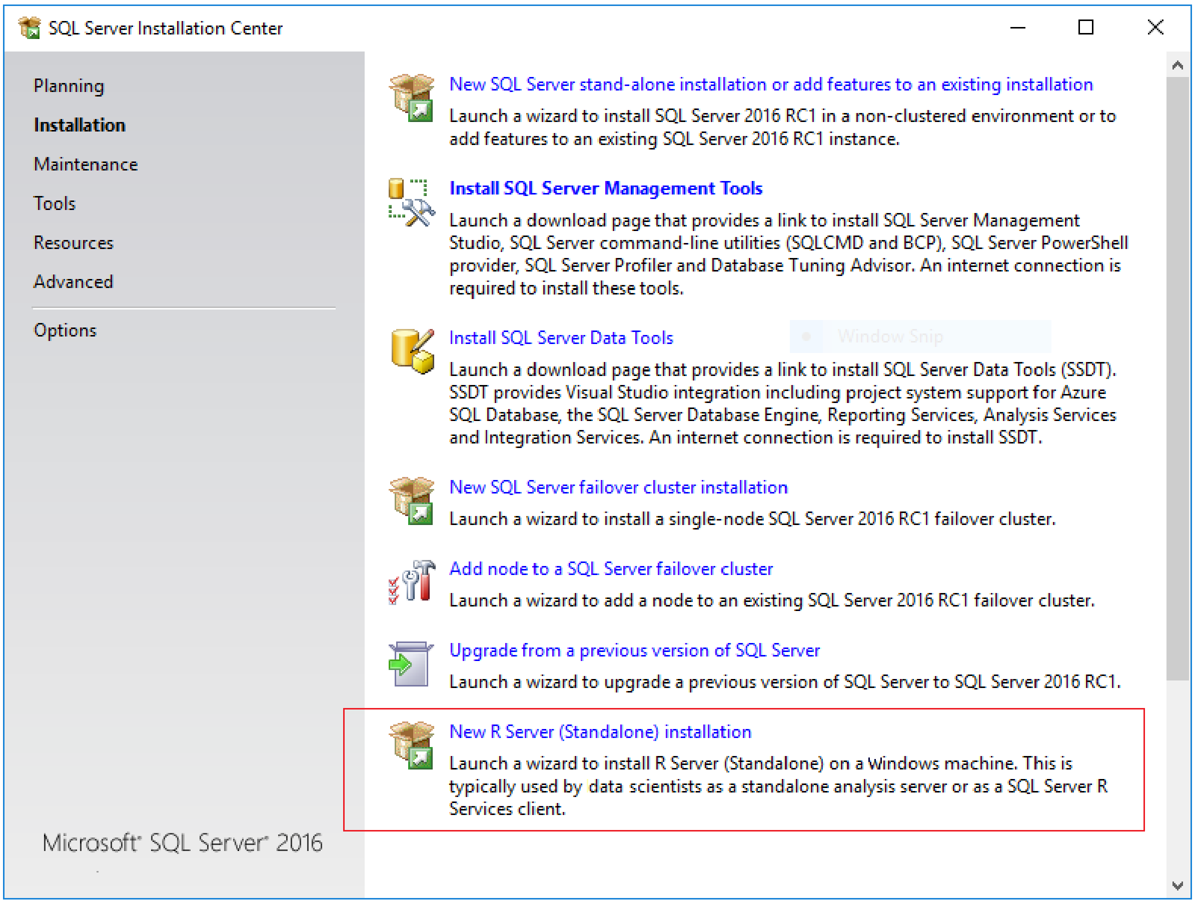
For more information about automated or off-line installation, see Install Microsoft R Server from the Command Line.

Install Microsoft R Server (Standalone)

1. If you have installed a previous version of Microsoft R Server, or any version of the Revolution Analytics tools, you must uninstall it first. See Upgrading from an Older Version of Microsoft R Server.

2. Run SQL Server 2016 setup. We recommend that you install Service Pack 1 or later.

3. On the **Installation** tab, click **New R Server (Standalone) installation** .



New SQL Server stand-alone installation or add features to an existing installation

Launch a wizard to install SQL Server 2016 RC1 in a non-clustered environment or to add features to an existing SQL Server 2016 RC1 instance.

Install SQL Server Management Tools

Launch a download page that provides a link to install SQL Server Management Studio, SQL Server command-line utilities (SQLCMD and BCP), SQL Server PowerShell provider, SQL Server Profiler and Database Tuning Advisor. An internet connection is required to install these tools.

Install SQL Server Data Tools

Launch a download page that provides a link to install SQL Server Data Tools (SSDT). SSDT provides Visual Studio integration including project system support for Azure SQL Database, the SQL Server Database Engine, Reporting Services, Analysis Services and Integration Services. An internet connection is required to install SSDT.

New SQL Server failover cluster installation

Launch a wizard to install a single-node SQL Server 2016 RC1 failover cluster.

Add node to a SQL Server failover cluster

Launch a wizard to add a node to an existing SQL Server 2016 RC1 failover cluster.

Upgrade from a previous version of SQL Server

Launch a wizard to upgrade a previous version of SQL Server to SQL Server 2016 RC1.

New R Server (Standalone) installation

Launch a wizard to install R Server (Standalone) on a Windows machine. This is typically used by data scientists as a standalone analysis server or as a SQL Server R Services client.

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

   R Server (Standalone)

   All other options can be ignored. Do not install the SQL Server databae engine or SQL Server R Services.

5. 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.

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

Upgrade an Existing R Server to 9.0.1

If you installed an earlier version of Microsoft R Server (Standalone) you can upgrade the instance to use newer versions of the R components. The upgrade also changes the server Modern Lifecycle Support policy. This allows the instance to be updated more frequently, on a different schedule than the SQL Server releases.

1. Install Microsoft R Server (Standalone), if it is not already installed.

2. Download the separate Windows-based installer from the locations listed here: Run Microsoft R Server for Windows.

3. Run the installer and follow these instructions.

Change in Default Folder for R Packages

When you install using SLq 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.

R Server (Standalone) with setup using SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\130\R_SERVER
```

R Server (Standalone) with setup using SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\140\R_SERVER
```

Setup using the Windows standalone installer

However, if you install using the separate Windows installer, or if you upgrade using the separate Windows installer, the R libraries are moved to the following R Server folder:

```
C:\Program Files\Microsoft\R Server\R_SERVER
```

Setup of R Servies or Machine LEarning Services In-Databases

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 into a different folder:

```
C:\Program Files\Microsoft SQL Server\<instance_name>\R_SERVICES
```

Do not directly call the R packages or utilities associated with the SQL Server instance. If both R Server and R Services are installed on the same computer, when you need to run RGui or other tools, use the R tools and packages installed by the R_SERVER folder.

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 the new release of R Tools for Visual Studio, as Visual Studio supports both R and Python, as well as SQL Server and BI tools. However, you can use any preferred development environment, including RStudio.

Troubleshooting

Incompatible version of R Client and R Server

If you install the latest version of Microsoft R Client and use it to run R on SQL Server using a remote compute context, you might get 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.

Typically, the version of R that is installed with SQL Server R Services is updated when service releases are published. To ensure that you always have the most up-to-date versions of R components, install all service packs. For compatibility with Microsoft R Client 9.0.0, you must install the updates that are described in this support article.

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 applied 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.

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.

To uninstall R Server (Standalone)

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.

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 or SQL Server 2016.

Unable 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

Microsoft R Server

Install Microsoft R Server from the Command Line

4/29/2017 • 3 min to read • Edit Online

This topic describes how to use SQL Server command-line arguments to install Microsoft R Server in SQL Server 2016, or install Machine Learning Server (Standalone) in SQL Server 2017.

> **NOTE**
>
> You can also install Microsoft R Server by using a separate Windows installer. For more information, see Install R Server 9.0.1 for Windows.

Prerequisites

This method of installation requires that you know how to perform a command-line installation of SQL Server and are familiar with its scripting arguments.

- **Unattended** installation requires that you specify the location of the setup utility, and use arguments to indicate which features to install.
- For a **quiet** installation, provide the same arguments and add the **/q** switch. No prompts will be provided and no interaction is required. However, setup will fail if any required arguments are omitted.

For more information, see Install SQL Server 2016 from the Command Prompt.

SQL Server 2017: Microsoft Machine Learning Server (Standalone)

Run the following command from an elevated command prompt to install only Microsoft Machine Learning Server (Standalone) and its prerequisites. The example shows the arguments used to install R.

```
Setup.exe /q /ACTION=Install /FEATURES=SQL_SHARED_MR, SQL_INST_MR  /IACCEPTROPENLICENSETERMS
/IACCEPTSQLSERVERLICENSETERMS
```

To view progress and prompts, remove the */q* argument.

- **FEATURES = SQL_SHARED_MR** gets only the Machine Learning Server components. This includes any prerequisites, which are installed silently by default.
- **SQL_INST_MR** is required to installl support for the R language.
- **SQL_INST_MPY** is required to install support for Python.
- **IACCEPTROPENLICENSETERMS** indicates you have accepted the license terms for using the open source R components.
- **IACCEPTPYTHONLICENSETERMS** indicates you have accepted the license terms for using the Python components.
- **IACCEPTSQLSERVERLICENSETERMS** is required to run the setup wizard.

Notes

1. The FEATURES argument is required, as is the SQL Server licensing terms.
2. Specify at least one language, together with the licensing agreement flag.
3. You can install one language, or both R and Python, but a separate license is required for each.

SQL Server 2016: Microsoft R Server (Standalone)

Run the following command from an elevated command prompt to install only Microsoft R Server (Standalone) and its prerequisites. The example shows the arguments used with SQL Server 2016 setup.

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

Offline Installation

If you install Machine Learning Server or Microsoft R Server (Standalone) on a computer that has no Internet access, you must download the required R components in advance, and copy them to a local folder. For download locations, see Installing R Components without Internet Access.

What Is Installed

After setup is complete, you can review the configuration file created by SQL Server setup, along with a summary of setup actions.

By default, all setup logs and summaries for SQL Server and related features are created in the following folders:

- SQL Server 2016: `C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log`
- SQL Server 2017: `C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log`

A separate subfolder is created for each feature installed.

To set up another instance of Microsoft R Server with the same parameters, you can re-use the configuration file that is created during installation. For more information, see Install SQL Server Using a Configuration File

Customize Your R Environment

After installation, you can install additional R packages. For more information, see Installing and Managing R Packages.

> **IMPORTANT**
>
> If you intend to run your R code on SQL Server, make sure that you install the same packages on the computer you'll use for developing the solution, and the SQL Server instance where you'll execute or deploy the solution.

After you have installed Machine Learning Services for R (In-Database), you can use the separate Windows installer to upgrade the version of R that is associated with a specified SQL Server instance. For more information, see Use SqlBindR to Upgrade R.

Advanced Analytics Virtual Machines on Azure

6/5/2017 • 5 min to read • Edit Online

Virtual machines on Azure are a convenient option for quickly configuring a complete server environment for machine learning solutions. This topic lists some virtual machine images that contain R Server, SQL Server with machine learning, or other data science tools from Microsoft .

> **TIP**
>
> We recommend that you use the new version of the Azure portal, and the Azure Marketplace. Some images are not available when browsing the Azure Gallery on the classic portal.

The Azure Marketplace contains multiple virtual machines that support data science. This list is not intended to be comprehensive, but only provide the names of images that include either Microsoft R Server or SQL Server machine learning services, to facilitate discovery.

How to Provision a VM

If you are new to using Azure VMs, we recommend that you see these articles for more information about using the portal and configuring a virtual machine.

- Virtual Machines - Getting Started
- Getting Started with Windows Virtual Machines

Find an image

1. From the Azure dashboard, click **Marketplace**.

   - Click **Intelligence and analytics** or **Databases**, and then type "R" in the **Filter** control to see a list of R Server virtual machines.
   - Other possible strings for the **Filter** control are *data science* and *machine learning*
   - Use the % wildcard in search to find VM names that contain a target string, such as *R* or *Julia*.

2. To get R Server for Windows, select **R Server Only SQL Server 2016 Enterprise**.

   R Server is licensed as a SQL Server Enterprise Edition feature, but version 9.1. is installed as a standalone server and serviced under the Modern Lifecycle support policy.

3. After the VM has been created and is running, click the **Connect** button to open a connection and log into the new machine.

4. After you connect, you might need to install additional R tools or development tools.

Install additional R tools

By default, Microsoft R Server includes all the R tools installed with a base installation of R, including RTerm and RGui. A shortcut to RGui has been added to the desktop, if you want to get started using R right away.

However, you might wish to install additional R tools, such as RStudio, the R Tools for Visual Studio (RTVS), or Microsoft R Client. See the following links for download locations and instructions:

- R Tools for Visual Studio
- Microsoft R Client
- RStudio for Windows

After installation is complete, be sure to change the default R runtime location so that all R development tools use the Microsoft R Server libraries.

Configure server for web services

If the virtual machine includes R Server, additional configuration is required to use web service deployment, remote execution, or leverage R Server as a deployment server in your organization. For instructions, see Configure R Server for Operationalization.

> **NOTE**
>
> Additional configuration is not needed if you just want to use packages such as RevoScaleR or MicrosoftML.

R Server Images

Microsoft Data Science Virtual Machine

Comes configured with Microsoft R Server, as well as Python (Anaconda distribution), a Jupyter notebook server, Visual Studio Community Edition, Power BI Desktop, the Azure SDK, and SQL Server Express edition.

The Windows version runs on Windows Server 2012, and contains many special tools for modeling and analytics, including CNTK and mxnet, popular R packages such as xgboost, and Vowpal Wabbit.

Linux Data Science Virtual Machine

Also contains popular tools for data science and development activities, including Microsoft R Open, Microsoft R Server Developer Edition, Anaconda Python, and Jupyter notebooks for Python, R and Julia.

This VM image was recently updated to include JupyterHub, open source software that enables use by multiple users, through different authentication methods, including local OS account authentication, and Github account authentication. JupyterHub is a particularly useful option if you want to run a training class and want all students to share the same server, but use separate notebooks and directories.

Images are provided for Ubuntu, Centos, and Centos CSP.

R Server Only SQL Server 2016 Enterprise

This virtual machine includes a standalone installer for R Server 9.1. that supports the new Modern Software Lifecycle licensing model.

R Server is also available in images for Linux CentOS version 7.2, Linux RedHat version 7.2, and Ubuntu version 16.04.

> **NOTE**
>
> These virtual machines replace the **RRE for Windows Virtual Machine** that was previously available in the Azure Marketplace.

SQL Server Images

To use R Services (In-Database) or Machine Learning Services, you must install one of the SQL Server Enterprise or Developer edition virtual machines, and add the machine learning service, as described here: Installing SQL Server R Services on an Azure Virtual Machine.

NOTE

Currently, machine learning services are not supported on the Linux virtual machines for SQL Server 2017, or in Azure SQL Database. You must use either SQL Server 2016 SP1 or SQL Server 2017 for Windows.

The Data Science Virtual Machine also includes SQL Server 2016 with the R services feature already enabled.

Other VMs

Deep Learning Toolkit for the Data Science Virtual Machine

This virtual machine contains the same machine learning tools available on the Data Science Virtual machine, but with GPU versions of mxnet, CNTK, TensorFlow, and Keras. This image can be used only on Azure GPU N-series instances.

The deep learning toolkit also provides a set of sample deep learning solutions that use the GPU, including image recognition on the CIFAR-10 database and a character recognition sample on the MNIST database. GPU instances are currently available in South Central US, East US, West Europe, and Southeast Asia.

IMPORTANT

GPU instances are currently only available in the South Central US.

Frequently Asked Questions

Can I install a virtual machine with SQL Server 2017?

Images are available that contain SQL Server 2017 CTP 2.0 for Linux environments, but these environments currently do not support Machine Learning Services.

A Windows-based virtual machine for SQL Server 2017 Enterprise Edition that includes Machine Learning Services will be available after public release.

As an alternative, you can use the SQL Server 2016 image, and upgrade your instance of R as described here: Upgrade an Instance using SqlBindR.

Or, create a virtual machine and download the CTP 2.0 preview of SQL Server 2017.

How do I access data in an Azure storage account?

When you need to use data from your Azure storage account, there are several options for accessing or moving the data:

- Copy the data from your storage account to the local file system using a utility, such as AzCopy.
- Add the files to a file share on your storage account and then mount the file share as a network drive on your VM. For more information, see Mounting Azure files.

How do I use data from Azure Data Lake Storage (ADLS)?

You can read data from ADLS storage using RevoScaleR, if you reference the storage account the same way that you would an HDFS file system, by using webHDFS. For more information, see this article: Using R to perform FileSystem Operations on Azure Data Lake Store.

See Also

SQL Server R Services

SQL Server Machine Learning Tasks

4/19/2017 • 4 min to read • Edit Online

R Services (In-Database) combines the power and flexibility of the open source R language with enterprise-level tools for data storage and management, workflow development, and reporting and visualization. This topic describes the machine learning lifecycle, and how SQL Server supports the needs of four different data professionals who are enagged in machine learning.

Machine Learning Life Cycle

Machine learning is not a short-term task, but rather a long-term process that touches all aspects of data in the enterprise. Machine learning begins with identification of business goals and rules, and collection of data from sensors and business applications. Machine learning is highly dependent on processes for extracting, processing, and storing data, and is increasingly important when considering policies for storing, extracting, and auditing data. Finally, machine learning is now an important omponent of strategies for reporting and analysis, as well as customer engagement and feedback.

SQL Server is an ideal fit for machine learning, because it bridges many of the gaps in the machine learning process:

- Works on-premises or in the cloud
- Integrated at every stage of enterprise data processing, including business intelligence
- Supports improved data security
- Provides resource governance and auditing

Data Professionals And How they Use Machine Learning

Data Scientists

Data scientists have access to 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, integration with SQL Server provides unique benefits:

- Develop and test your solutions by using the R development environment of your choice.
- Push computations to the database, avoiding data movement while complying 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.
- R code can be easily deployed to production and called by enterprise tools, applications, other databases, and dashboards.
- Data scientists can deploy and update an analytical solution while meeting standard requirements for enterprise data management, including security and access auditing
- Code portability: easily re-use your R code against other data sources, such as Hadoop

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 and deploy solutions.

Integration with SQL Server provides these benefits to data developers who work with machine learning:

- Use familiar tools to interact with R scripts. Let the data scientist work in RStudio while the data developer deploys the solution using SQL Server Management Studio. No more recoding of R or Python solutions.
- Optimize by mixing and matching SQL and R, or SQL and Python. Many times, complex operations on large datasets can be run far more efficiently using SQL Server features, such as in-memory columnstoreindexes, or very fast aggregates in T-SQL. Use a machine learning language where it makes sense, and use SQL to move and process data.
- Effortlessly automate tasks that must run repeatedly on large amounts of data, such as generating prediction scores on production data.
- Execute R or Python script from any application that uses Transact-SQL. Just call a stored procedure to create a parameterized model, generate a complex plot, or output predictions.
- The **RevoScaleR** and **revoscalepy** APIs can operate on 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 who can install new R packages, using database roles.
- 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 R Solutions

Architects and ETL Designers

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 tht are part of the machine learning process. Often, the overall data platform must be designed to balance competing and complementary 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.

- Familiar development tools for developng R and Python solutions. You can 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 transform and manage data, and use cloud data sources in machine learning workflows.

- Scheduling and management using operationalization features in Microsoft R Server.

For information on related tasks, see:

- Creating Workflows that Use R in SQL Server

R Package - Management for SQL Server R Services

4/29/2017 • 6 min to read • Edit Online

This topic describes the package management functions that you can use to manage R packages that are running on an instance of SQL Server.

The **RevoScaleR** package now includes functions to support easier installation and management of R packages. These new functions, combined with database roles for package management, supports these scenarios:

- The data scientist can install needed R packages on SQL Server without having administrative access to the SQL Server computer.
- The packages are installed on a per database basis, and when the database is moved, the packages move with it.
- It is easier to share packages with others. If you set up a local package repository, each data scientist can use the respository to install packages to his or her own database.
- The database administrator does not need to learn how to run R commands, and doesn't need to track complex package dependencies.
- The DBA can use familiar database roles to control which SQL Server users are permitted to install, uninstall, or use packages.

How Package Management Works

The database administrator is responsible for setting up roles and adding users to the roles, to control who has permission to add or remove R packages from the SQL Server environment.

If you have permission to install packages, you run one of the package management functions from your R code and specify the compute context where packages are to be added or removed. The compute context can be your local computer or a database on the SQL Server instance. If the call to install packages is run on SQL Server, your credentials determine whether the operation can be completed on the server. The package installation functions check for dependencies and ensure that any related packages can be installed to SQL Server, just like R package installation in the local compute context. The function that uninstalls packages also computes dependencies and ensures that packages that are no longer used by other packages on SQL Server are removed, to free up resources.

Each data scientist can install private packages that are not visible to others, creating a private sandbox for R packages. Because packages can be scoped to a database and each user gets an isolated package sandbox in each database, it is easier to install different versions of the same R package.

If you migrate your working database to a new server, you can use the package synchronization function to read a list of all your packages and install them in a database on the new server.

Supported Versions

- The R functions for package management are provided beginning with Microsoft R Server 9.0.1.
- These packages are included by default in SQL Server 2017.
- You can add the packages to an instance of SQL Server 2016 if you upgrade the instance to use at least Microsoft R 9.0.1. For more information, see Using SqlBindR.exe to Upgrade R.

Database roles and database scoping

The new package management functions provide two scopes for installation and use of packages in SQL Server on a particular database:

- **Shared scope**

  Shared scope means that users who have been given permission to the shared scope role (**rpkgs-shared**) can install and uninstall packages to a specified database. A package that is installed in a shared scope library can be used by other users of the database on SQL Server, provided those users are allowed to use installed R packages.

- **Private scope**

  Private scope means that users who have been given membership in the private scope role (**rpkgs-private**) can install or uninstall packages into a private library location defined per user. Therefore, any packages installed in the private scope can be used only by the user who installed them. In other words, a user on SQL Server cannot use private packages that were installed by a different user.

These models for *shared* and *private* scope can be combined to develop custom secure systems for deploying and managing packages on SQL Server.

For example, by using shared scope, the lead or manager for a group of data scientists could be granted permission to install packages, and those packages could then be used by all other users or data scientists in the same SQL Server instance.

Another scenario might require greater isolation among users, or use of different versions of packages. In that case, private scope can be used to give individual permissions to data scientists, who would be responsible for installing and using just the packages they need. Because packages are installed on a per-user basis, packages installed by one user would not affect the work of other users who are using the same SQL Server database.

Database roles for package management

The following new database roles support secure installation and R package management in SQL Server:

- **rpkgs-users** Allows users to use any shared packages that were installed by members of the **rpkgs-shared** role.

- **rpkgs-private** Provides access to shared packages with the same permissions as the **rpkgs-users** role. Members of this role can also install, remove and use privately scoped packages.

- **rpkgs-shared** Provides the same permissions as the **rpkgs-private** role. Users who are members of this role can also install or remove shared packages.

- **db_owner** - Has the same permissions as the **rpkgs-shared** role. Can also grant users the right to install or remove both shared and private packages.

R Package Synchronization

The CTP 2.0 release of SQ Server 2017 (and the April 2017 release of Microsoft R Server) includes new R functions for *synchronizing packages*.

Package synchronization means that the database engine tracks the packages that are used by a specific owner and group, and can write those packages to the file system if needed. You can use package synchronization in these scenarios:

- You want to move R packages between instances of SQL Server
- You need to re-install packages for a specific user or group after a database is restored

For more information, see rxSyncPackages.

List of Package Management Functions

- `rxInstalledPackages`: Find information about packages installed in the specified compute context.

- `rxInstallPackages` : Install packages into a compute context, either from a specified repository, or by reading locally saved zipped packages.
- `rxRemovePackages` : Remove installed packages from a compute context.
- `rxFindPackage` : Get the path for one or more packages in the specified compute context.
- `rxSqlLibPaths` : Get the search path for the library trees for packages while executing inside the SQL Server.

Examples

Get package location on SQL Server compute context

This example gets the path for the **RevoScaleR** package on the compute context, *sqlServer*.

```
sqlPackagePaths <- rxFindPackage(package = "RevoScaleR", computeContext = sqlServerL)
```

Get locations for multiple packages

The following example gets the paths for the **RevoScaleR** and **lattice** packages, on the compute context, *sqlServer*. When finding information about multiple packages, pass a string vector containing the package names.

```
packagePaths <- rxFindPackage(package = c("RevoScaleR", "lattice"), computeContext = sqlServer)
```

List packages in specified compute context

This example lists and then displays in the console all packages installed in the compute context, *sqlServer*.

```
myPackages <- rxInstalledPackages(computeContext = sqlServer)
myPackages
```

Get package versions

This example gets the build number and version numbers for a package installed on the compute context, *sqlServer*.

```
sqlPackages <- rxInstalledPackages(fields = c("Package", "Version", "Built"), computeContext = sqlServer)
```

Install a package on SQL Server

This example installs the **ggplot2** package and its dependencies into the compute context, *sqlServer*.

```
pkgs <- c("ggplot2")
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlServer)
```

Remove a package from SQL Server

This example removes the **ggplot2** package and its dependencies from the compute context, *sqlServer*.

```
pkgs <- c("ggplot2")
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlServer)
```

See Also

How to Enable or Disable R Package Management

R Package - How to Enable or Disable

4/19/2017 • 1 min to read • Edit Online

By default, package management is disabled on a SQL Server instance, even if R Services is installed. To enable this feature is a two-step process which must be performed by a database administrator:

1. Enable package management on the SQL Server instance (once per SQL Server instance)
2. Enable package management on the SQL database (once per SQL Server database)

When disabling the package management feature, reverse the process to remove database-level packages and permissions, and then remove the roles from the server:

1. Disable package management on each database (once per database)
2. Disable package management on the SQL Server instance (once per instance)

> **IMPORTANT**
>
> This feature is in development. Please be aware that the syntax or functionality might change in later releases.

To enable package management

To enable or disable package management requires the command-line utility **RegisterRExt.exe**, which is included with the **RevoScaleR** package installed with SQL Server R Services. the default location is:

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExe.exe
```

1. Open an elevated command prompt and use the following command:

   ```
   RegisterRExt.exe /installpkgmgmt [/instance:name] [/user:username] [/password:*|password]
   ```

   This command creates instance-level artifacts on the SQL Server computer that are required for package management.

2. To add package management at the database level, for each database where packages must be installed, run the following command from an elevated command prompt:

   ```
   RegisterRExt.exe /installpkgmgmt /database:databasename [/instance:name] [/user:username]
   [/password:*|password]
   ```

   This command creates some database artifacts, including the following database roles that are required for controlling user permissions: **rpkgs-users**, **rpkgs-private**, and **rpkgs-shared**

To disable package management

1. From an elevated command prompt, run the following command to disable package management at the database level:

   ```
   RegisterRExt.exe /uninstallpkgmgmt /database:databasename [/instance:name] [/user:username]
   [/password:*|password]
   ```

   This command will remove database artifacts related to package management from the specified database. The command will also remove all the packages that were installed per database from the secured file system location on the SQL Server computer.

   The command must be run once for each database where package management was used.

2. (Optional) To entirely remove the package management feature from the instance, 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:name] [/user:username] [/password:*|password]
```

This command removes the instance-level artifacts used by package management from the SQL Server instance.

See Also

R Package Management for SQL Server R Services

Installing and Managing R Packages

4/19/2017 • 4 min to read • Edit Online

Any R solution that runs in R Services (In-Database) must use packages that are installed in the default R library. More typically, R solutions will reference user libraries by specifying a file path in the R code, but this is not recommended for production.

Therefore, it is the task of the database administrator or other administrator on the server to ensure that all required packages are installed on the SQL Server instance. If you do not have administrative privileges on the computer that hosts the SQL Server instance, you can provide to the administrator information about how to install R packages, and provide access to a secure package repository where packages requested by users can be obtained. This section provides that information.

TIP

SQL Server 2017 provides new features for installing and managing R packages that give both the database administrator and the data scientist greater freedom and control over package usage and setup. For more information, see R Package Management for SQL Server.

Installed Packages

When you install R Services (In-Database), by default the R **base** packages are installed, such as `stats` and `utils`, together with the **RevoScaleR** package that supports connections to SQL Server.

NOTE

For a list of packages installed by default, see Packages Installed with Microsoft R Open.

If you need an additional package from CRAN or another repository, you must download the package and install it on your workstation.

If you need to run the new package in the context of the server, an administrator must install it on the server as well.

Where and How to Get New Packages

There are multiple sources for R packages, the best known among them being CRAN and Bioconductor. The official site for the R language (https://www.r-project.org/) lists many of these resources. Many packages are also published to GitHub, where you can obtain the source code. However, you may also have been given R packages that were developed by someone in your company.

Regardless of the source, packages must be provided in the format of a zipped file to be installed. Moreover, to use the package with R Services (In-Database), be sure to get the zipped file in the Windows binary format. (Some packages might not support this format.) For more information about the contents of the zip file format, and how to create an R package, we recommend this tutorial, which you can download in PDF format from the R project site: Freidrich Leisch: Creating R Packages.

In general, R packages can be installed easily from the command line without downloading them in advance, if the computer has Internet access. Generally this is not the case with servers running SQL Server . Therefore, to install an R package to a computer that does **not** have Internet access, you must download the package in the correct zipped format ahead of time and then copy the zipped files to a folder that is accessible by the computer.

The following topics describe two methods for installing packages off-line:

- Create an Offline Package Repository using miniCRAN

  Describes how to use the R package **miniCRAN** to create an off-line repository. This is probably the most efficient method if you need to install packages to multiple servers and manage the repository from a single location.

- Install New R Packages from the Internet

  Includes instructions for installing packages off-line by manually copying zipped files.

Permissions Required for Installing R Packages

To install a new R package on a computer that is running SQL Server 2017, you must have administrative rights to the computer.

If you do not have these rights, contact your administrator and provide the information about the package to install.

If you are installing a new R package on a computer that is being used as an R workstation and the computer does **not** have an instance of SQL Server installed, you still need administrative rights to the computer to install the package. After you have installed the package, you can run it locally.

> **NOTE**
>
> In SQL Server 2017, new database roles have been added to support scoping of package installation permissions at an instance level and database level. For more information, see R Package Management for SQL Server.

Location of default R library location for R Services

If you installed R Services (In-Database) using the default instance, the R package library used by the instance is located under the SQL Server instance folder. For example:

- Default instance *MSSQLSERVER* `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library`
- Named instance *MyNamedInstance* `C:\Program Files\Microsoft SQL Server\MSSQL13.MyNamedInstance\R_SERVICES\library`

You can run the following statement to verify the default library for the current instance of R.

```
EXECUTE sp_execute_external_script  @language = N'R'
, @script = N'OutputDataSet <- data.frame(.libPaths());'
WITH RESULT SETS (([DefaultLibraryName] VARCHAR(MAX) NOT NULL));
GO
```

For more information, see Determine Which Packages are Installed on SQL Server.

Managing Installed Packages

SQL Server 2017 provides new features for installing and managing R packages that give both the database administrator and the data scientist greater freedom and control over package usage and setup. For more information, see R Package Management for SQL Server.

If you are using SQL Server 2106 R Services, the new package management features are not available at this time. In the meantime, you have these options for determining which packages are installed on the SQL Server computer, use one of these options:

- View the default library, if you have permissions to the folder.
- Run a command from R command to list the packages in the R_SERVICES library location
- Use a stored procedure such as the following on the instance:

  ```
  SQL EXECUTE sp_execute_external_script @language=N'R' ,@script = N'str(OutputDataSet); packagematrix <-
  installed.packages(); NameOnly <- packagematrix[,1]; OutputDataSet <- as.data.frame(NameOnly);'
  ,@input_data_1 = N'SELECT 1 as col' WITH RESULT SETS ((PackageName nvarchar(250) ))
  ```

See Also

Managing and Monitoring R Solutions

Install Additional R Packages on SQL Server

4/19/2017 • 4 min to read • Edit Online

This topic describes how to install new R packages to an instance of R Services (In-Database) on a computer that has access to the Internet.

1. Locate the Windows binaries in ZIP file format

R packages are supported on many platforms. You must ensure that the package you want to install has a binary format for the Windows platform. Otherwise the downloaded package will not work.

For example, to obtain the FISHalyseR package from Bioconductor:

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. You can then install the package, or copy the zipped package to a server that does not have Internet access.

2. Open the default R package library for SQL Server R Services

Navigate to the folder on the server where the R packages associated with R Services (In-Database) have been installed. It is important that you install packages to the default library that is associated with the current instance.

For instructions on how to locate this library, see Installing and Managing R Packages.

For each instance where you will run a package, you must install a separate copy of the packages. Currently packages cannot be shared across instances.

3. Open an administrative command prompt

Open R as an administrator. You can do this by using the Windows command p,ropt, or by using one of the R utilities.

Using the Windows command prompt

1. Open a Windows command prompt as administrator, and navigate to the directory where the RTerm.Exe or RGui.exe files are located.

   In a default install, this is the R **\bin** directory. For example, in SQL Server 2017, the R tools are located here:

   Default instance

   ```
   C:\Program Files\MSSQL13.MSSQLSERVER\R_SERVICES\bin
   ```

   Named instance

   ```
   C:\Program files\MSSQL13.<instanceName>\R_SERVICES\bin\x64
   ```

2. Run **R.Exe**.

Using the R command-line utilities

1. Use Windows Explorer to navigate to the directory containing the R tools.

2. Right-click **RGui.exe** or **RTerm.exe**, and select **Run as administrator**.

4. Install the package

The R command to install the package depends on whether you are getting the package from the Internet or from a local zipped file.

Install package from Internet

1. In general, you use the following command to install a new package from CRAN or one of the mirror sites.

   ```
   install.packages("target_package_name")
   ```

   Note that double quotation marks are always required for the package name.

2. To specify the library where the package should be installed, use a command like this one to set the library location:

   ```
   lib.SQL <- "C:\\Program Files\\Microsoft SQL Server\\MSSQL13.MSSQLSERVER\\R_SERVICES\\library"
   ```

   Note that for R Services (In-Database), currently only one package library is allowed. Do not install packages to a user library, or you will not be able to run the package from SQL Server.

3. Having defined the library location, the following statement installs the popular e1070 package into the package library used by R Services.

   ```
   install.packages("e1071", lib = lib.SQL)
   ```

4. You will be asked for a mirror site from which to obtain the package. Select any mirror site that is convenient for your location.

   For a list of current CRAN mirrors, see this site.

   > **TIP**
   >
   > To avoid having to select a mirror site each time you add a new package, you can configure your R development environment to always use the same repository.
   >
   > To do this, edit the global R settings file, .Rprofile, and add the following line:
   >
   > `options(repos=structure(c(CRAN="<mirror site URL>")))`
   >
   > For detailed information about preferences and other files loaded when the R runtime starts, run this command from an R console:
   >
   > `?Startup`

5. If the target package depends on additional packages, the R installer will automatically download the dependencies and install them for you.

Manual package installation, or installing on computer with no Internet access

1. If the package that you intend to install has dependencies, get the required packages ahead of time and add them to the folder with other package zipped files.

TIP

We recommend that you set up a local repository using miniCRAN if you need to support frequent offline installation of R packages.

2. At the R command prompt, type the following command to specify the path and name of the package to install:

   ```
   install.packages("C:\\Temp\\Downloaded packages\\mynewpackage.zip", repos=NULL)
   ```

   This command extracts the R package from its local zipped file, assuming you saved the copy in the directory `C:\Temp\Downloaded packages`, and installs the package (with its dependencies) into the R library on the local computer.

3. If you have previously modified the R environment on the computer, you should ensure that the R environment variable `.libPath` uses just one path, which points to the R_SERVICES folder for the instance.

NOTE

If you have installed Microsoft R Server (Standalone) in addition to SQL Server R Services, your computer will have a separate installation of R with all the R tools and libraries. Packages that are installed to the R_SERVER library are used only by Microsoft R Server and cannot be accessed by SQL Server.

Be sure to use the R_SERVICES library when installing packages that you want to use in SQL Server.

See Also

Set up SQL Server R Services (In-Database)

Create a Local Package Repository Using miniCRAN

4/29/2017 • 3 min to read • Edit Online

This topic describes how you can create a local R package repository using the R package **miniCRAN**.

Because SQL Server instances typically are located on a server that does not have Internet connectivity, the standard method of installing R packages (the R command `install.packages()`) might not work, as the package installer cannot access CRAN or any other mirror sites.

There are two options for installing packages from a local share or repository:

- Use the miniCRAN package to create a local repository of the packages you need, then install from this repository. This topic describes the miniCRAN method.

- Download the packages you need, and their dependencies, as zip files, and save them in a local folder, and then copy that folder to the SQL Server computer. For more information on the manual copy method, see Install Additional Packages on SQL Server.

Step 1. Install miniCRAN and download packages

1. Install the miniCRAN package on a computer that has Internet access.

```
# Install miniCRAN and igraph

if(!require("miniCRAN")) install.packages("miniCRAN")
if(!require("igraph")) install.packages("igraph")
library(miniCRAN)

# Define the package source: a CRAN mirror, or an MRAN snapshot
CRAN_mirror <- c(CRAN = "https://mran.microsoft.com/snapshot/2016-04-01")

# Define the local download location
local_repo <- "~/miniCRAN"
```

2. Download or install the packages you need to this computer usng the following R script. This will create the folder structure that you need to copy the packages to the SQL Server later.

```
# List the packages to get. Do not specify dependencies.
pkgs_needed <- c("ggplot2", "ggdendro")
# Plot the dependency graph
plot(makeDepGraph(pkgs_needed))

# Create the local repo
pkgs_expanded <- pkgDep(pkgs_needed, repos = CRAN_mirror)
makeRepo(pkgs_expanded, path = local_repo, repos = CRAN_mirror, type = "win.binary", Rversion = "3.2")

# List local packages
pdb <- as.data.frame(
  pkgAvail(local_repo, type = "win.binary", Rversion = "3.2"),
  stringsAsFactors = FALSE)
head(pdb)
pdb$Package
pdb[, c("Package", "Version", "License")]
```

Step 2. Copy the miniCRAN repository to the SQL Server computer

Copy the miniCRAN repository to the R_SERVICES library on the SQL Server instance.

- For SQL Server 2016, the default folder is `C:/Program Files/Microsoft SQL Server/MSSQL13.MSSQLSERVER/R_SERVICES/library1`.
- For SQL Server 2017, the default folder is `C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library1`.

If you have installed R Services using a named instance, be sure to include the instance name in the path, to ensure that the libraries are installed to the correct instance. For example, if your named instance is RTEST02, the default path for the named instance would be: `C:\Program Files\Microsoft SQL Server\MSSQL13.RTEST02\R_SERVICES\library`.

If you have installed SQL Server to a different drive, or made any other changes in the installation path, be sure to make those changes as well.

Step 3. Install the packages on SQL Server using the miniCRAN repository

On the SQL Server computer, open an R command line or RGUI as administrator.

TIP

You might have multiple R libraries on the computer; therefore, to ensure that packages are installed to the correct instance, use the copy of RGUI or RTerm that is installed with the specific instance where you want to install the packages.

When prompted to specify a repository, select the folder containing the files you just copied; that is, the local miniCRAN repository.

```
# Run this R code as administrator on the SQL Server computer
pkgs_needed <- c("ggplot2", "ggdendro")
local_repo  <- "~/miniCRAN"

# OPTIONAL: If you are not running R from the instance library as recommended, you must specify the path
#   .libPaths()[1]
# "C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library "
# lib <- .libPaths()[1]

install.packages(pkgs_needed,
                 repos = file.path("file://", normalizePath(local_repo, winslash = "/")),
                 lib = lib,
                 type = "win.binary",
                 dependencies = TRUE
                 )
installed.packages()
```

Verify that the packages were installed.

```
installed.packages()
```

Acknowledgements

The source for this information is this article by Andre de Vries, who also developed the miniCRAN package. For details and a complete walkthrough, see How to install R packages on an off-line SQL Server 2016 instance

Determine Which Packages are Installed on SQL Server

5/15/2017 • 1 min to read • Edit Online

This topic describes how you can determine which R packages are installed on the SQL Server instance.

By default, installation of R Services (In-Database) creates an R package library associated with each instance. Therefore, to know which packages are installed on a computer, you must run this query on each separate instance where R Services is installed. Note that package libraries are **not** shared across instances, so it is possible for different packages to be installed on different instances.

For information about how to determine the default library location for an instance, see Installing and Managing R Packages.

Get a List of Installed Packages Using R

There are multiple ways to get a list of installed or loaded packages using R tools and R functions.

- Many R development tools provide an object browser or a list of packages that are installed or that are loaded in the current R workspace.
- We recommend the following functions from the RevoScaleR package that are provided specifically for package management in compute contexts:
  - rxFindPackage
  - rxInstalledPackages
- You can use an R function, such as `installed.packages()`, which is included in the installed `utils` package. The function scans the DESCRIPTION files of each package that was found in the specified library and returns a matrix of package names, library paths, and version numbers.

Examples

The following example uses the function `rxInstalledPackages` to get a list of packages available in the provided SQL Server compute context.

```
sqlServerCompute <- RxInSqlServer(connectionString =
"Driver=SQL Server;Server=myServer;Database=TestDB;Uid=myID;Pwd=myPwd;")
     sqlPackages <- rxInstalledPackages(computeContext = sqlServerCompute)
     sqlPackages
```

The following example uses the base 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. To avoid parsing the 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();
NameOnly <- packagematrix[,1];
OutputDataSet <- as.data.frame(NameOnly);'
,@input_data_1 = N'SELECT 1 as col'
WITH RESULT SETS ((PackageName nvarchar(250) ))
```

For more information, see the description of optional and default fields for the R package DESCRIPTION file, at https://cran.r-project.org.

See Also

Install Additional R Packages on SQL Server

Packages Installed in User Libraries

4/19/2017 • 1 min to read • Edit Online

If a user needs a new R package and is not an administrator, the packages cannot be installed to the default location and are by default installed into a private, user library.

In a typical R development environment, to reference the package in code, the user would have to add the location to the R environment variable `libPath`, or reference the full package path, like this:

```
library("c:/Users/<username>/R/win-library/packagename")
```

Problems with packages in user libraries

However, in R Services (In-Database), this workaround is **not** supported; packages must be installed to the default library. If the package is not installed in the default library, you might get this error when you try to call the package:

Error in library(xxx) : there is no package called 'xxx'

Therefore, when you migrate R solutions to run in SQL Server, it is important that you do the following:

- Install any packages that you need to the default library.
- Edit code to ensure that packages are loaded from the default library, not from ad hoc directories or user libraries.
- Check your code to make sure that there are no calls to uninstalled packages.
- Modify any code that would try to install packages dynamically.

If a package is installed in the default library, the R runtime will load the package from the default library, even if a different library is specified in the R code.

See Also

R Package Synchronization for SQL Server

4/29/2017 • 6 min to read • Edit Online

SQL Server 2017 CTP 2.0 includes a new function for synchronization of R packages, to support back up and restore of R package collections associated with SQL Server databases. This feature helps ensure that complex sets of R packages created by users are not lost and can be easily restored.

This topic describes what the package synchronization feature does, and how to use the `rxSyncPackages()` function to perform the following tasks:

- Synchronize a list of packages for an entire SQL Server database
- Synchonize packages used by an individual user, or by a group of users

> **NOTE**
>
> This function is provided as a part of pre-release software and is subject to change before final release.

What is Package Synchronization

Package synchronization is a new feature that works specifically with SQL Server compute contexts. It is designed to get a list of R packages that are installed on a particular database, for a particular user or group, and ensure that the packages listed in the file system match those in the database.

This is useful if you need to move a user database and move the packages along with the database. You can also use package synchronization when you back up and restore a SQL Server database used for R jobs.

Package synchronization uses a new function, `rxSyncPackages()`. To synchronize the list of packages, you open an R command prompt, pass the compute context that defines the instance and database you want to work with, and then provide either a package scope or a user or owner name.

How packages are managed in R and SQL Server

Typically, when you run R scripts using standard R tools, R packages are installed on the file system. If multiple people use R on the same computer, there might be many copies of the same packages, in different folders or in diffrent user libraries.

However, to use an R package from SQL Server, the package must be installed in the default R library that is associated with the instance. A server computer might be hosting multiple instances of SQL Server with R enabled, and in this case, each instance can have a separate set of R packages.

The database administrator is responsible for installing packages on the instance. However, with the package management libraries, the administrator can delegate this responsibility to users.

- For each database, the administrator can give users the ability to freely install the R packages they need. This mechanism ensures that multiple users can install different versions of R packages without causing conflicts for other users of the SQL Server computer. Individual users can install packages for their own use, using a file system location marked as **private**, if they belong to the database role **rpkgs-private**.

- The administrator can set up a group of package users on a database, and install packages that are shared by all users in the group. Packages can be shared among members of the database role **rpkgs-shared**. Such users can also install packages to private scope locations.

Goal of package synchronization

If a database on a server is lost or must be moved, by using package synchronization, you can restore sets of packages specific to a database, user, or group.

The information about users and the packages that they have installed is stored in the SQL Server instance, and is used to update the packages in the file system. Whenever you add a new package using the package management functions, both the records in SQL Server and the file system are updated. Therefore, if a user moves to a different SQL Server, you can take a backup of the user's working databaes 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.

Supported Versions

This function is included in SQL Server 2017 CTP 2.0.

Because this function is part of Microsoft R version 9.1.0, you can add this feature to a 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.

To Synchronize Packages

You call `rxSyncPackages` after restoring an instance of SQL Server to a new machine, or if a R package on the file system is believed to be corrupted.

If the command executes successfully, existing packages in the file system are added to the database, scope, and owner as specified. If the file system is corrupted, the packages are restred based on the list maintained in the database.

Syntax

```
rxSyncPackages(computeContext = rxGetOption("computeContext"), scope = c("shared", "private"), owner = c(),
verbose = getOption("verbose"))
```

- Compute context

  Define a SQL Server compute context, consisting of an instance and database, and the packages to sync. Create the SQ Server context by using the `RxInSqlServer` function. If you don't specify a compute context, the current compute context is used.

- Scope

  Indicate whether you are installing packages for a single user, or for a group of users:

  - **private** The operation will include only those packages that have been installed for use by a specified owner.
  - **shared** The oepration will include all packages installed for a group of users.

  If you run the function without specifying either private or shared scope, both scopes are applied. As a result, the entire set of packages available for all scopes and users will be copied.

- Owner

  Specify the owner of the packages to synchronize. The owner name must be a valid SQL database user. If you leave it empty, the user name of SQL login specified in the connection is used.

Requirements

- The person who executes the function must be a security principal on the SQL Server instance and database that has the packages, and must be a member of a package management role: **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 sychronize packages marked as **private**, either the owner of the package or the administrator must

run the function, and the packages must be private.

- **rpkgs-users** -Members of this role can run code that uses packages installed on the SQL Server instance, but cannot install or synch packages.
- To sync packages on behalf of other users, the owner must be a member of the **db_owner** database role.

Examples

The following examples create a connection to a specific instance of SQL Server, specify a database, and then specify a set of packages to synchronize.

When the call to `rxSyncPackages` is made, the package lists are synchronized between the file system and the database.

Synchronize all by database

This example gets all packages installed 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)
```

Restrict synchronized packages by scope

The following examples synchonize only the packages in either shared scope or private scope.

Shared scope

```
rxSyncPackages(computeContext=computeContext, scope="shared", verbose=TRUE)
```

Private scope

```
rxSyncPackages(computeContext=computeContext, scope="private", verbose=TRUE)
```

Restrict synchronized packages by owner

The following example demonstrates how to get 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))
```

See Also

R Package Management for SQL Server

R Libraries and R Data Types

4/29/2017 • 7 min to read • Edit Online

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 the R Server Release Notes.

Additional information about recent releases can be found in the R Server - What's New notes.

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
bigint	`numeric`	**float**	

SQL TYPE	R CLASS	RESULT SET TYPE	COMMENTS
binary(n) n <= 8000	`raw`	**varbinary(max)**	Only allowed as input parameter and output
bit	`logical`	**bit**	
char(n) n <= 8000	`character`	**varchar(max)**	
datetime	`POSIXct`	**datetime**	Represented as GMT
date	`POSIXct`	**datetime**	Represented as GMT
decimal(p,s)	`numeric`	**float**	
float	`numeric`	**float**	
int	`integer`	**int**	
money	`numeric`	**float**	
numeric(p,s)	`numeric`	**float**	
real	`numeric`	**float**	
smalldatetime	`POSIXct`	**datetime**	Represented as GMT
smallint	`integer`	**int**	
smallmoney	`numeric`	**float**	
tinyint	`integer`	**int**	
uniqueidentifier	`character`	**varchar(max)**	
varbinary(n) n <= 8000	`raw`	**varbinary(max)**	Only allowed as input parameter and output
varbinary(max)	`raw`	**varbinary(max)**	Only allowed as input parameter and output
varchar(n) n <= 8000	`character`	**varchar(max)**	

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 informatioh 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 *varsToDrop* 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

4/29/2017 • 1 min to read • Edit Online

This topic describes the Python libraries that are included with the following products:

- SQL Server Machine Learning Services (In-Database)
- Microsoft Machine Learning Server (Standalone)

This topic 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 packages 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
bigint	`numeric`
binary	`raw`
bit	`bool`
char	`str`
float	`float64`
int	`int32`
nchar	`str`
nvarchar	`str`
nvarchar(max)	`str`

SQLTYPE	PYTHON TYPE
real	`float32`
smallint	`int16`
tinyint	`uint8`
varbinary	`bytes`
varbinary(max)	`bytes`
varchar(n)	`str`
varchar(max)	`str`

Real-time Scoring

4/29/2017 • 7 min to read • Edit Online

This topic describes a new feature in SQL Server 2017 that provides scoring on machine learning models in near real-time.

- What is real-time scoring vs. native scoring
- How it works
- Supported platforms

What is Real-Time 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 that pairs R with SQL Server means that an external R processes must be invoked for every call, regardless of the complexity of the operation. In cases where you are only 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.

In large deployments, scoring more typically involve getting scores from a model and inserting the scores into a table for many millions of rows, or retrieving a single score in real time. To support extremely fast execution of scoring in these scenarios, SQL Server Machine Learning Services (and Microsoft Machine Learning Server) now provide built-in prediction and scoring libraries.

Supported platforms

- Microsoft R Server
- SQL Server R Services 2016: requires upgrade of the R Services instance to Microsoft R Server 9.1.0
- SQL Server Machine Learning Services (2017; includes Microsoft R Server 9.1.0)

Benefits

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. This supports much faster prediction performance in enterprise production scenarios.

Also, 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.

How it Works

To perform real-time scoring in SQL Server, you call sp_rxPredict, a new stored procedure provided specifically for real-time scoring.

In Microsoft R Server (or Machine Learning Server Standalone), the equivalent functionality is provided by the rxPredict function.

This real-time scoring library has been designed in C+ and optimized specifically for scoring operations. With sp_rxPredict, you can load a specified model from a database, define a query with new input data, and generate scores based on the model. You can run the scoring library either from SQL Server, using the stored procedure, or from R code, by calling the rxPredict function.

Some restrictions apply:

- On SQL Server, you must enable the real-time scoring feature in advance. This is because the feature entails installation of CLR-based libraries into SQL Server.
- The model must be trained in advance and saved using a new serialization function provided in Microsoft R Server 9.1.0. The serialization function is optimized to support fast scoring.
- The model must be *published* to SQL Server or another supported platform.
- Only some model types can be used for scoring. Currently a selection of models from the MicrosoftML and revoScaleR pacakges are supported.
- Models containing R transformations are not supported currently.

See the Technical Notes section for a list of further restrictions and known issues.

Perform Real-Time Scoring

1. Enable the serialization feature
2. Serialize the model using the format required for real-time scoring
3. Publish the model to SQL Server as described here
4. Use T-SQL to call real-time scoring stored procedure

Step 1. Enable serialization on the database

By default, real-time scoring functionality is disabled in SQL Server. You must enable this feature for each SQL Server database that you want to use for scoring.

> **NOTE**
>
> In order for real-time scoring to work, SQL CLR functionality needs to be enabled in the instance and the database needs to be marked trustworthy. Please carefully consider additional security implications of doing this.

A server administrator must use the RegisterRExt.exe command-line utility to enable real-time scoring. This utility is included with the RevoScaleR package.

To enable real-time scoring

1. Open an elevated command prompt, and locate RegisterRExt.exe. In a default installation, the path is `<SQLInstancePath>\R_SERVICES\library\RevoScaleR\x64\`
2. Run the following command, substituting the name of your instance and the target database:

   ```
   RegisterRExt.exe /installRts [/instance:name] /database:databasename
   ```

   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 will run the real-time scoring functionality.

To disable real-time scoring functionality

1. Open an elevated command prompt.
2. Run the following command:

   ```
   RegisterRExt.exe /uninstallrts /database:databasename [/instance:name]
   ```

Step 2. Serialize the model

To provide the best performance, real-time scoring includes a new serialization method for storing models and loading them for scoring. The serialization format uses an optimized raw format (not human readable) that allows the model to be scored in an efficient manner.

When you train a model, call the new serialization function to save the model in a SQL Server table using the new format that supports real-time scoring.

- `rxSerializeModel()` Serializes a supported model in raw format.

  The model must be based on a supported model type from the RevoScaleR or MicrosoftML packages. Only models saved with this format can be loaded into SQL Server for real-time scoring.

- `rxUnserializeModel()` Reconstitutes the original R model object from the serialized raw format.

Step 3. Publish model to SQL Server

Publishing refers to the process of saving a trained model, using the new serialized format, in a table in SQL Server. A model that has been published using this process can be read from a table for real-time scoring, without having to invoke the R runtime.

Requirements

- The database containing the table must have real-time scoring enabled.
- Use a column of type VARBINARY to save the serialized models.

When creating the model from your 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* flag in `rxWriteObject` to FALSE, to avoid repeating the serialization step.

- Save the model in raw format to a file and then read from the file into SQL Server.

Step 4. Call sp_rxPredict

After you have enabled the real-time scoring feature, the SQL Server database will have a new stored procedure called **sp_rxPredict**.

Syntax

```
sp_rxPredict @model = @model,
@inputData = N'SELECT * FROM data'
```

- *@model* A trained model to be used for scoring. The model must have been trained using one of the supported algorithms, and saved using the new `rxSerializeModel()` function.

- *@inputData* A valid SQL query that defines the input data for scoring. The data must include columns compatible with the columns in the stored model.

- Currently, `sp_rxPredict` supports only the following .NET column types: double, float, short, ushort, long, ulong and string. You may need to filter out unsupported types in your input data before using it for real-time scoring.

  For information about corresponding SQL types, see SQL-CLR Type Mapping or Mapping CLR Parameter Data.

See the Technical Notes section for a list of further restrictions and known issues.

Supported algorithms

- RevoScaleR models

  `rxLinMod` , `rxLogit` , `rxDTree` , `rxDForest` and `rxBTrees`

- Microsoft ML models

  `rxFastTrees` , `rxFastForest` , `rxLogisticRegression` , `rxOneClassSvm` , `rxNeuralNet` , `rxFastLinear` , `featurizeText` , `concat` , `categorical` , `categoricalHash` , `selectFeatures`

Algorithms and models not supported

- Models containing R transformations
- Models using the `rxGlm` or `rxNaiveBayes` algorithms in RevoScaleR
- PMML models
- Models created using other R libraries from CRAN or other repositories

Real-Time Scoring in Microsoft R Server

For information regarding real-time 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 real-time scoring as a new a web service running on R Server.

Technical Notes

Real-time scoring does not use an R interpreter; therefore, any functionality that might require R interpreter is not supported during real-time scoring. 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 real-time 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 real-time scoring.

Known Issues

- `sp_rxPredict` is optimized for fast predictions on smaller data sets, ranging from a few rows to a few thousand rows. Performance might begin to degrading as data sets grow larger.

- `sp_rxPredict` returns an inaccurate message when NULL value is passed as model System.Data.SqlTypes.SqlNullValueException:Data in Null

See Also

Data Exploration and Predictive Modeling with R

4/29/2017 • 5 min to read • Edit Online

This topic 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 Learnign 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 hae 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 has 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 Get started with ScaleR and data analysis](https://msdn.microsoft.com/microsoft-r/scaler-getting-started).

- **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 alorithms:

- **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

Save and Load R Objects from SQL Server using ODBC

4/19/2017 • 3 min to read • Edit Online

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. Moreover, you can save R objects to SQL Server directly from an R environment, by calling these new functions over an ODBC connection using *RxOdbcData*.

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. These functions require that a connection be established to SQL Server using the *RxOdbcData* data source.

In general, the function calls are modeled after 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.

- 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.
- Optionally, you can specify that the object not be serialized, and you can choose a new compression algorithm to use instead of the default compression algorithm.

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 will be available in the ScaleR reference on MSDN at a later date.

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")
```

See Also

R Libraries and R Data Types

4/29/2017 • 7 min to read • Edit Online

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 the R Server Release Notes.

Additional information about recent releases can be found in the R Server - What's New notes.

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
bigint	`numeric`	**float**	

SQL TYPE	R CLASS	RESULT SET TYPE	COMMENTS
binary(n) n <= 8000	`raw`	**varbinary(max)**	Only allowed as input parameter and output
bit	`logical`	**bit**	
char(n) n <= 8000	`character`	**varchar(max)**	
datetime	`POSIXct`	**datetime**	Represented as GMT
date	`POSIXct`	**datetime**	Represented as GMT
decimal(p,s)	`numeric`	**float**	
float	`numeric`	**float**	
int	`integer`	**int**	
money	`numeric`	**float**	
numeric(p,s)	`numeric`	**float**	
real	`numeric`	**float**	
smalldatetime	`POSIXct`	**datetime**	Represented as GMT
smallint	`integer`	**int**	
smallmoney	`numeric`	**float**	
tinyint	`integer`	**int**	
uniqueidentifier	`character`	**varchar(max)**	
varbinary(n) n <= 8000	`raw`	**varbinary(max)**	Only allowed as input parameter and output
varbinary(max)	`raw`	**varbinary(max)**	Only allowed as input parameter and output
varchar(n) n <= 8000	`character`	**varchar(max)**	

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 informatioh 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 *varsToDrop* 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

Converting R Code for Use in R Services

4/19/2017 • 5 min to read • Edit Online

When you move R code from R Studio or another environment to SQL Server, often the code will work without further modification when added to the *@script* parameter of sp_execute_external_script. This is particularly true if you have developed your solution using the ScaleR functions, making it relatively simple to change execution contexts.

However, typically you will want to modify your R code to run in SQL Server, both to take advantage of the tighter integration with SQL Server, and to avoid expensive transfer of data.

Resources

To view examples of how R code can be run in SQL Server, see these walkthroughs:

- 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

Changes to the Data Science Process in SQL Server

When you are working in R Tools for Visual Studio, RStudio, or other environment, a typical workflow is to 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:

EXTERNAL CODE	R IN SQL SERVER
Ingest data	Define input data as a SQL query inside database
Summarize and visualize data	No change; plots can be exported as images or sent to a remote workstation
Feature engineering	You can use R in-database, but it can be more efficient to call T-SQL functions or custom UDFs
Data cleansing part of analysis process	Data cleansing can be delegated to ETL processes and performed in advance
Output predictions to file	Output predictions to table or wrap `predict` in stored procedures for direct access by applications

Best Practices

- Make a note of dependencies, such as required packages, in advance. In a development and testing environment, it might be okay to install packages as part of your code, but this is a bad practice in a production environment. Notify the administrator so that packages can be installed and tested in advance of deploying your code.
- Make a checklist of possible data type issues. Document the schemas of the results expected in each section of the code.

- Identify primary data sources such as model training data, or input data for predictions vs. secondary sources such as factors, additional grouping variables, and so forth. Map your largest dataset to the input parameter of sp_execute_external_script.
- Change your input data statements to work directly in the database. Rather than moving data to a local CSV file, or making repeated ODBC calls, you'll get better performance by using SQL queries or views that can be run directly against the database, avoiding data movement.
- Use 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.
- Where possible, replace conventional R functions with **ScaleR** functions that support distributed execution. For more information, see Comparison of Functions in Scale R and CRAN R.
- 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 decide to perform feature engineering or feature extraction separately and add the values in a new column. Use T-SQL rather than R code for set-based computations. For one example of an R solution that compares performance of UDFs and R for feature engineering tasks, see this tutorial: Data Science End-to-End Walkthrough.

Restrictions

Keep in mind the following restrictions when converting your R code:

Data types

- All R data types are supported. However, SQL Server supports a greater variety of data types than does R, so some implicit data type conversions are performed when sending SQL Server data to R and vice versa. You might also need to explicitly cast or covert some data.
- NULL values are supported. R uses the `na` data construct to represent missing values, which is similar to nulls.
- For more information, see Working with R Data Types.

  Inputs and outputs

- You can define only one input dataset as part of the stored procedure parameters. This input query for the stored procedure can be any valid single SELECT statement. We recommend that you use this input for the biggest dataset, and get smaller datasets if needed by using calls to RODBC.
- Stored procedure calls preceded by EXECUTE cannot be used as an input to sp_execute_external_script.
- All columns in the input dataset must be mapped to variables in the R script. Variables are mapped automatically by name. For example, say your R script contains a formula like this one:

  ```
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
  ```

  An error will be raised if the input dataset does not contains columns with the matching names ArrDelay, CRSDepTime, DayOfWeek, CRSDepHour, and DayOfWeek.
- You can also provide multiple scalar parameters as input. However, 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.
- To include scalar input variables in the output of the R code, just append the **OUTPUT** keyword when you define

the variable.

- In R Services (In-Database), your R code is limited to outputting a single dataset as a data.frame object. However, you can also output multiple scalar outputs, including plots in binary format, and models in varbinary format.

- You can usually output the dataset returned by the R script without having to specify names of the output columns, by using the WITH RESULT SETS UNDEFINED option. However, any variables in the R script that you want to output must be mapped to SQL output parameters.

- If your R script uses the argument `@parallel=1`, you must define the output schema. 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 ouput schema must be defined before the parallel processes can be created.

  Dependencies

  - Avoid installing packages from R code. On SQL Server, packages should be installed in advance, and you must use the default package library used by R Services.

Creating multiple models using rxExecBy

4/29/2017 • 3 min to read • Edit Online

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 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 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 thee 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 AirlineDemoSmall]'
, @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

4/19/2017 • 3 min to read • Edit Online

The **olapR** package is a new R package, provided in Microsoft R Server and SQL Server R Services, that lets you run MDX queries and use the data from OLAP cubes in your R solution.

With this package, you don't need to create linked servers or clean up flattened rowsets; you can use OLAP data directly in R,

Overview

An OLAP cube is a multi-dimensional database that contains precalculated aggregations over *measures*, which typically capture important business metrics such as sales amount, sales count, or other numeric values. OLAP cubes 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

The **olapR** package supports two methods of creating MDX queries:

- You can generate a simple MDX query by using an R-style API to choose a cube, axes, and slicers. Using these function, you can build valid MDX queries even if you do not have access to traditional OLAP tools, or don't have deep knowledge of the MDX language.

  Note that not all possible MDX queries can be created by using this method in the **olapR** package, as MDX can be very complex. However, it supports all of the most common and useful operations, including slice, dice, drilldown, rollup, and pivot in N dimensions.

- You can manually create and then paste in any MDX query. This option is useful 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.

  With this approach, you build your MDX using any client utility, such as SSMS or Excel, and then use it as a string argument to the *SSAS query handler* that is provided by this package. The **olapR** function will send the query to the specified Analysis Services server, and pass 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.

MDX Basics

Data in a cube can be retrieved using the MDX (MultiDimensional Expression) query language. Because the data in an OLAP cube (or Analysis Services database) is multidimensional rather than tabular, MDX supports a complex syntax and a variety of operations for filtering and slicing data:

- *Slicing* takes a subset of the cube by picking a value for one dimension, resulting in a cube that is one dimension smaller.
- *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.

This topic provides more examples following examples show the basic syntax for querying a cube. How to Create MDX Queries using R

Known Issues

Tabular models not supported currently

You can connect to a tabular instance of Analysis Services and the `explore` function will report success with a return value of TRUE, but the tabular model objects are not a compatible type and cannot be explored.

Tabular models support MDX queries, but a valid MDX query against a tabular model will return a NULL result and not report an error.

Resources

If you are new to OLAP or to MDX queries, see these Wikipedia articles: OLAP Cubes MDX queries

Samples

If you want to learn more about cubes, you can create the cube that is used in these examples by following the Analysis Services tutorial up to Lesson 4: Creating an OLAP Cube

You can also download an existing cube as a backup, and restore it to an instance of Analysis Services. For example, you can download a fully processed cube for Adventure Works Multidimensional Model SQL 2014, in zipped format, and restore it to your SSAS instance. For more information, see Backup and Restore, or Restore-ASDatabase Cmdlet.

See Also

How to Create MDX Queries using R

MDX Query Designer for Analysis Services

Create a Stored Procedure Using sqlrutils

4/29/2017 • 7 min to read • Edit Online

This topic 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 topic.

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_rsample** 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_rsample, 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 `registerStoredProcedure` function in **sqlrutils** to register the stored procedure with the database.

  For example, you could register the stored procedure **sp_rsample** in the instance and database defined in *sqlConnStr*, by making this R call:

  ```
  registerStoredProcedure(sp_rsample, 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 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 `executeStoredProcedure` 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

SQL Server R Services Performance Tuning

4/19/2017 • 1 min to read • Edit Online

R Services (In-Database) provides a platform for developing intelligent applications that uncover new insights. A data scientist can use the power of R language to train and create models using data stored inside SQL Server. Once the model is ready for production, a data scientist can work with database administrators and SQL engineers to deploy their solution in production. The information in this section provides high level guidance on tuning performance both when creating and training models, and when deploying models to production.

The information in this document assumes that you are familiar with R Services (In-Database) concepts and terminology. For general information on R Services, see SQL Server R Services.

> **NOTE**
>
> While much of the information in this section is general guidance on configuring SQL Server, some information is specific to RevoScaleR analytic functions.

In This Section

- SQL Server Configuration: This document provides guidance for configuring the hardware that SQL Server is installed on. It is most useful to **Database Administrators**.
- R and Data Optimization: This document provides guidance on using R scripts with R Services. It is most useful to **Data Scientists**.
- Performance Case Study: This document provides test data and R scripts that can be used to test the impact of guidance provided in the previous documents.

References

The following are links to information used in the development of this document.

- How to determine the appropriate page file size for 64-bit versions of Windows
- Data Compression
- Enable Compression on a Table or Index
- Disable Compression on a Table or Index
- DISKSPD storage load generator/performance test tool
- FSUtil utility reference
- Reorganize and Rebuild Indexes
- R Services
- Performance Tuning Options for rxDForest and rxDTree
- Monitor and Tune for Performance
- Explore R and ScaleR in 25 Functions
- Resource Governor

See Also

SQL Server Configuration for R Services

Performance Case Study

R and Data Optimization

SQL Server Configuration (R Services)

4/19/2017 • 4 min to read • Edit Online

The information in this section provides general guidance on the hardware and network configuration of the computer that is used to host R Services (In-Database). It should be considered in addition to the general SQL Server performance tuning information provided in the Performance Center for SQL Server Database Engine and Azure SQL Database.

Processor

R Services (In-Database) can perform tasks in parallel by using the available cores on the machine; the more cores that are available, the better the performance. Since SQL Server is normally used by multiple users simultaneously, the database administrator should determine the ideal number of cores that are needed to support peak workload computations. While the number of cores may not help for IO bound operations, CPU bound algorithms will benefit from faster CPUs with many cores.

Memory

The amount of memory available on the computer can have a large impact on the performance of advanced analytic algorithms. Insufficient memory may 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 32GB is highly recommended. If you have more than 32GB available, you can configure the SQL data source to use more rows in every read operation to improve performance.

Power Options

On the Windows operating system, the **High Performance** power option should be used. Using a different power setting will result in decreased or inconsistent performance when using R Services (In-Database).

Disk IO

Training and prediction jobs using R Services (In-Database) 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 SQL Server 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**
>
> R Services (In-Database) requires 8.3 filename support on Windows operating systems. You can use fsutil.exe to determine whether a drive supports 8.3 filenames, or to enable support if it does not. For more information on using fsutil.exe with 8.3 filenames, see Fsutil 8dot3name.

Table Compression

IO performance can often be improved by using either compression or columstore indexes. Generally, data is often repeated in several columns within a table, so using a columnstore index takes advantage of these repetitions when compressing the data.

A columnstore index might not be as efficient if there are a lot of 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

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.

See How to determine the appropriate page file size for 64-bit versions of Windows for information on sizing the page file.

Resource Governance

SQL Server supports resource governance for controlling the various resources used by R Services (In-Database). For example, the default value for memory consumption by R is limited to 20% of the total memory available for SQL Server. This is done to ensure that SQL Server workflows are not severely affected by long running R jobs. However, these limits can be changed by the database administrator.

The resources limited are **MAX_CPU_PERCENT**, **MAX_MEMORY_PERCENT**, and **MAX_PROCESSES**. To view the current settings, use this Transact-SQL statement:

```
SELECT * FROM sys.resource_governor_external_resource_pools
```

If SQL Server is primarily used for R Services, it might be helpful to increase MAX_CPU_PERCENT to 40% or 60%. If there many R sessions using the same SQL Server at the same time, all three will be increased. To change the allocated resource values, use Transact-SQL statements.

This example sets the memory usage to 40%:

```
ALTER EXTERNAL RESOURCE POOL [default] WITH (MAX_MEMORY_PERCENT = 40)
```

The following example sets all three configurable values:

```
ALTER EXTERNAL RESOURCE POOL [default] WITH (MAX_CPU_PERCENT = 40, MAX_MEMORY_PERCENT = 50, MAX_PROCESSES =
20)`
```

NOTE

To make changes to these settings take effect immediately, run the statement `ALTER RESOURCE GOVERNOR RECONFIGURE` after changing a memory, CPU, or max process setting.

See Also

Resource Governor

CREATE EXTERNAL RESOURCE POOL

SQL Server R Services Performance Tuning Guide

Performance Case Study

R and Data Optimization

R and Data Optimization (R Services)

4/19/2017 • 8 min to read • Edit Online

This topic describes methods for updating your R code to improve performance or avoid known issues.

Compute Context

R Services (In-Database) can use either the **local** or **SQL** compute context when performing analysis. When using the **local** compute context, analysis is performed on the client machine and data must be fetched from SQL Server 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.

If the compute context is **SQL Server**, then the analytic functions are executed inside SQL Server. The data is local to the analysis task, so no network overhead is introduced.

When working with large data sets, you should always use the SQL compute context.

Factors

The R language converts strings from tables into factors. Many data source objects take `colInfo` as a parameter to control how the columns are treated. For example, `c(“fruit” = c(type = “factor”, levels=as.character(c(1:3)), newLevels=c(“apple”, “orange”, “banana”)))` will consume integers 1, 2, and 3 from a table and treat them as factors with levels `apple`, `orange`, and `banana`.

Data scientists often use factor variables in their formula; however, using factors when the source data is an integer will incur a performance hit as integers are converted to strings at run time. However, if the column contains strings, you can specify the levels ahead of time using `colInfo`. In this case, the equivalent statement would be `c(“fruit” = c(type = “factor”, levels= c(“apple”, “orange”, “banana”)))`, which treats the strings as factors as they are being read.

To avoid run time conversions, consider storing levels as integers in the table and consuming them as described in the first formula example. If there is no semantic difference in the model generation, then this approach can lead to better performance.

Data Transformation

Data scientists often use transformation functions written in R as part of the analysis. The transformation functions must be applied to each row retrieved from the table. In R Services (In-Database), this transformation happens in batch mode and involves 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. Therefore, using transformations can havea 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, as this avoids 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.

Batching

The SQL data source (`RxSqlServerData`) has an option to indicate the batch size using the parameter `rowsPerRead`. This parameter specifies the number of rows to process at a time. At run time, algorithms will read the specified numbered of rows in each batch. By default, the value of this parameter is set to 50,000, to ensure that the algorithms can perform well even on machines with low memory. If the machine has enough available memory, increasing this value to 500,000 or even a million can yield better performance, especially for large tables.

Increasing this value may not always produce better results and may require some experimentation to determine the optimal value. The benefits of this will be more evident on a large data set with multiple processes (`numTasks` set to a value greater than `1`).

Parallel Processing

To improve the performance of running rx analytic functions inside SQL Server, R Services (In-Database) relies on parallel processing using the available cores on the SQL Server machine. There are two ways to achieve parallelization with R Services (In-Database):

- When using the `sp_execute_external_script` stored procedure to run an R script, set the `@parallel` parameter to `1`. This is useful for R scripts that do not use RevoScaleR functions, which are generally prefixed with "rx". If the script uses RevoScaleR functions, parallel processing is handled automatically and you should not set `@parallel` to `1`.

  If the R script can be parallelized, and if the Transact-SQL query can be parallelized, then SQL Server will create multiple parallel processes (up to the **max degree of parallelism MAXDOP** setting for SQL Server,) and run the same script across all processes. Each process only receives a portion of the data, so this 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.

- When using rx functions with a SQL Server compute context, set `numTasks` to the number of processes you wish to create. The actual number of processes created is determined by SQL Server, and may be less than you requested. The number of processes created can never be more than **MAXDOP**.

  If the R script can be parallelized, and if the Transact-SQL query can be parallelized, then SQL Server will create multiple parallel processes when running the rx functions.

The number of processes that will be 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

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 it can render itself for parallel execution.

R Services (In-Database) supports working with SQL data sources using `RxSqlServerData` to specify the source. The source can be either a table or a query. For example, the following code samples both define an R data source object based on a SQL query:

```
RxSqlServerData(table=”airline”, connectionString = sqlConnString)
```

```
RxSqlServerData(sqlquery=”select [ArrDelay],[CRSDepTime],[DayOfWeek] from airlineWithIndex where rowNum <=
100000”, connectionString = sqlConnString)
```

As the analytics algorithms pull large volumes of data from the tables, it is important to ensure 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.

SQL Server Management Studio can be used to analyze the execution plan, and improve the performance of the query. For example, a missing index on a table can affect the time taken to execute a query. See Monitor and Tune for Performance for more information.

Another oversight that can affect the performance is when the query retrieves more columns than are required. For example, if a formula is based on only 3 columns, and the table has 30 columns, do not use a query such as `select *` or one that selects more columns than needed.

> **NOTE**
>
> If a table is specified in the data source instead of a query, R Services (In-Database) will internally determine the necessary columns to fetch from the table; however, this approach is unlikely to result in parallel execution.

Algorithm Parameters

Many rx training algorithms support parameters to control how the training model is generated. While the accuracy and correctness of the model is important, the performance of the algorithm may be equally important. You can modify parametersthe model training parameters to increase the speed of computation, and in many cases, you might be able to improve performance without reducing the accuracy or correctness.

For example, `rxDTree` supports the `maxDepth` parameter, which controls the maximum tree depth. As `maxDepth` is increased, performance can degrade, so it is important to analyze the benefits of increasing the depth vs. the performance impact.

One of the parameters that can be used with `rxLinMod` and `rxLogit` is the `cube` argument. This argument can be used when the first dependent variable of the formula is a factor variable. If `cube` is set to `TRUE`, the regression is done using a partitioned inverse, it may 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.

The RevoScaleR users guide has some useful information for controlling the model fit for various algorithms. For example, with `rxDTree` you can 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.

For more information on tuning performance for `rxDForest` and `rxDTree`, see Performance tuning options for rxDForest/rxDTree.

Model and Prediction

Once the training has completed and the best model selected, we recommend storing the model in the database so that it is readily available for predictions. For on-line transaction processing that requires prediction, loading the pre-computed model from the database for the prediction is very efficient. The sample scripts use this approach to serialize and store the model in a database table. For prediction, the model is de-serialized from the database.

Some models generated by algorithms such as lm or glm can be quite large, especially when used on a large data set. There are size limitations to the data that can be stored in SQL Server. You should clean up the model before storing it to the database.

Operationalization using Microsoft R Server

If fast prediction using a stored model and integrating the analytics into an application is an important scenario, you can also use the **operationalization** features (formerly known as DeployR) in Microsoft R Server.

- Data scientists can use the **mrsdeploy** package to share R code with other computers, and integrate R analytics inside web, desktop, mobile, and dashboard applications. For more information, see Getting Started for Data Scientists.
- Administrators can manage packages, monitor compute and web notes, and control security on R jobs. For

more information, see Getting Started for Administrators.

See Also

Resource Governance Resource Governor

CREATE EXTERNAL RESOURCE POOL

SQL Server R Services Performance Tuning Guide

SQL Server Configuration for R Services

Performance Case Study

Performance Case Study (R Services)

4/19/2017 • 7 min to read • Edit Online

To demonstrate the effect of the guidance provided in the previous sections, tests were run using the tables from the Airline data set.

Tests and Example Data

There are six tables, with 10M rows in each table:

TABLE NAME	DESCRIPTION
airline	Data converted from original xdf file using `rxDataStep`
airlineWithIntCol	*DayOfWeek* converted to an integer rather than a string. Also adds a *rowNum* column.
airlineWithIndex	The same data as the *airlineWithIntCol* table, but with a single clustered index using the *rowNum* column.
airlineWithPageComp	The same data as the *airlineWithIndex* table, but with page compression enabled. Also adds two columns, *CRSDepHour* and *Late*, which are computed from *CRSDepTime* and *ArrDelay*.
airlineWithRowComp	The same data as the *airlineWithIndex* table, but with row compression enabled. Also adds two columns, *CRSDepHour* and *Late* which are computed from *CRSDepTime* and *ArrDelay*.
airlineColumnar	A columnar store with a single clustered index. This table is populated with data from a cleaned up csv file.

The scripts used to perform the tests described in this section, as well as links to the example data used for the tests, are available at https://github.com/Microsoft/SQL-Server-R-Services-Samples/tree/master/PerfTuning.

Each test was run six times, and the time of the first run (the "cold run") was dropped. To allow for the occasional outlier, the maximum time for 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. R garbage collection was induced before each test. The value of *rowsPerRead* for each test was set to 500000.

Data Size When Using Compression and a Columnar Store Table

The following are the results of using compression and a columnar table to reduce the size of the data:

TABLE NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED	% SAVING (RESERVED)
airlineWithIndex	10000000	2978816 KB	2972160 KB	6128 KB	528 KB	0

TABLE NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED	% SAVING (RESERVED)
airlineWithPageComp	10000000	625784 KB	623744 KB	1352 KB	688 KB	79%
airlineWithRowComp	10000000	1262520 KB	1258880 KB	2552 KB	1088 KB	58%
airlineColumnar	9999999	201992 KB	201624 KB	n/a	368 KB	93%

Using Integer vs. String in Formula

In this experiment, `rxLinMod` was used with the two tables, one with string factors, and one with integer factors.

- For the *airline* table

  DayOfWeek is a string

  The `colInfo` parameter was used to specify the factor levels (`Monday`, `Tuesday`, ...)

- For the *airlineWithIndex* table

  DayOfWeek is an integer

  `colInfo` was not specified

In both cases, the same formula was used: `ArrDelay ~ CRSDepTime + DayOfWeek`.

The following results clearly show the benefit of using integers rather than strings for factors:

TABLE NAME	TEST NAME	AVERAGE TIME
airline	*FactorCol*	10.72
airlineWithIntCol	*IntCol*	3.4475

Using Compression

In this experiment, `rxLinMod` was used with multiple data tables: *airlineWithIndex*, *airlineWithPageComp*, and *airlineWithRowComp*. The same formula and query was used for all tables.

TABLE NAME	TEST NAME	NUMTASKS	AVERAGE TIME
airlineWithIndex	NoCompression	1	5.6775
		4	5.1775
airlineWithPageComp	PageCompression	1	6.7875
		4	5.3225
airlineWithRowComp	RowCompression	1	6.1325

TABLE NAME	TEST NAME	NUMTASKS	AVERAGE TIME
		4	5.2375

Note that compression alone (*numTasks* set to 1) does not seem to help in this example, as 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.

Avoiding Transformation Function

In this experiment, `rxLinMod` is used with the table *airlineWithIndex* in two runs, one using a transformation function, and one without the transformation function.

TEST NAME	AVERAGE TIME
WithTransformation	5.1675
WithoutTransformation	4.7

Note that there is improvement in time when not using a transformation function; in other words, when using columns that are pre-computed and persisted in the table. The savings would be much greater if there were many more transformations and the data set were larger (> 100M).

Using Columnar Store

In this experiment, `rxLinMod` was used with two tables, *airlineWithIndex* and *airlineColumnar*, and no transformation was used. These results indicate that the columnar store can perform better than row store. There will be a significant difference in performance on larger data set (> 100 M).

TABLE NAME	TEST NAME	AVERAGE TIME
airlineWithIndex	RowStore	4.67
airlineColumnar	ColStore	4.555

Effect of the Cube Parameter

In this experiment, `rxLinMod` is used with the *airline* table, in which the column *DayOfWeek* is stored as a string. The formula used is `ArrDelay ~ Origin:DayOfWeek + Month + DayofMonth + CRSDepTime`. The results clearly show that the use of the `cube` parameter helps with performance.

TEST NAME	CUBE PARAMETER	NUMTASKS	AVERAGE TIME	ONE ROW PREDICT (ARRDELAY_PRED)
CubeArgEffect	`cube = F`	1	91.0725	9.959204
		4	44.09	9.959204
	`cube = T`	1	21.1125	9.959204

TEST NAME	CUBE PARAMETER	NUMTASKS	AVERAGE TIME	ONE ROW PREDICT (ARRDELAY_PRED)
		4	8.08	9.959204

Effect of maxDepth for rxDTree

In this experiment, `rxDTree` is used with the *airlineColumnar* table. Several different values for *maxDepth* were used to demonstrate how it affects the run time complexity.

TEST NAME	MAXDEPTH	AVERAGE TIME
TreeDepthEffect	1	10.1975
	2	13.2575
	4	19.27
	8	45.5775
	16	339.54

As the depth increases, the total number of nodes increases exponentially and the elapsed time will increase significantly. For this test *numTasks* was set to 4.

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 6 times, and was performed twice under both power options to demonstrate the variability of results when using the Balanced power option. The results show that the numbers are more consistent and faster when using the high performance power plan.

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
	1	3.45 seconds	

TEST NAME	RUN #	ELAPSED TIME	AVERAGE TIME
	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
	1	3.82 seconds	
	2	3.84 seconds	
	3	3.86 seconds	
	4	4.07 seconds	
	5	4.86 seconds	
	6	3.75 seconds	
			3.88

Prediction Using a Stored Model

In this experiment, a model is created and stored to a database. Then the stored model is loaded from the database and predictions created using a one row data frame in memory (local compute context). The time taken to train, save, and load the model and predict is shown below. .

TABLE NAME	TEST NAME	AVERAGE TIME (TO TRAIN MODEL)	TIME TO SAVE/LOAD MODEL
airline	SaveModel	21.59	2.08
	LoadModelAndPredict		2.09 (includes time to predict)

The test results show the time to save the model and the time taken to load the model and predict. This is clearly a faster way to do prediction.

Performance Troubleshooting

The tests used in this section produce output files for each run by using the *reportProgress* parameter, which is passed to the tests with value `3`. The console output is directed to a file in the output directory. The output file contains information regarding the time spent in IO, transition time, and compute time. These times are useful for troubleshooting and diagnosis.

The test scripts then process these times for the various runs to come up with the average time over runs. For example, the following shows the sample times for a run. The main timings of interest are **Total read time** (IO time) and **Transition time** (overhead in setting up processes for computation).

```
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
```

TIP

These test scripts and techniques can also be useful in troubleshooting issues when using rx analytic functions on your SQL Server.

Scripts and Resources

- Github: Sample data and scripts for this case study
- Blog: Reference Implementation of Credit Risk Prediction using R

  A performance case study that includes downloadable source code.

- Monitoring R Services using Custom Reports

  Custom reports that can be viewed in SQL Server Management Studio.

See Also

SQL Server R Services Performance Tuning Guide

SQL Server Configuration for R Services

R and Data Optimization

Using R Code Profiling Functions

4/19/2017 • 1 min to read • Edit Online

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

Configuration and Management

4/29/2017 • 1 min to read • Edit Online

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 topics describe how to install new R packages on the SQL Server instance, manage R package libraries, and restore package libraries after a database restore.

- Installing and Managing R Packages
- 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 topics 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 topics describe how to implement resource management for R or Python jobs using the Resource Governor feature avaialble 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 topics:

- Upgrade and Installation FAQ
- Security Considerations
- Known Issues for R Services

Advanced Configuration Options for Machine Learning Services

4/29/2017 • 4 min to read • Edit Online

This article describes changes that you can make after setup, to modify the configuration of the R 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 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 the R runtime process. You can modify the number of users if needed to support R. Your database administrator must also give this group permission to connect to any instance where R Services has been enabled. For more information, see Modify the User Account Pool for SQL Server R Services.

However, an access control list (ACL) can be defined for sensitive resources on the SQL Server to deny access to this group to prevent the R runtime process from getting access to the resources.

- The user account pool is linked to a specific instance. For each instance on which R script has been enabled, a separate pool of worker accounts are created. Accounts cannot be shared between instances.
- User account names in the pool are of the format SQLInstanceName*nn*. For example, if you are using the default instance as your R server, 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 R runtime sessions that can be launched simultaneously is limited by the size of this user account pool. However, this limit can be changed by an administrator by using 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 R Services.

Manage Memory Used by External Script Processes

By default, the R runtime processes associated with R Services (In-Database) are limited to using no more than 20% of total machine memory. However, this limit can be increased by the administrator, if needed.

Generally, this amount will be inadequate for serious R tasks such as training model or predicting on many rows of data. You might need to reduce the amount of memory reserved for SQL Server (or for other services) and use Resource Governor to define an external resource pool or pools and allocate. For more information, see Resource Governance for R.

Change Advanced Service Options using the Configuration File

You can control some advanced properties of R Services (In-Database) by editing the R Services (In-Database) configuration file. This 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.

For example, to use Notepad to open the configuration file for the default instance, you would open a command prompt as administrator, and type the following command:

```
C:\>Notepad.exe "%programfiles%\Microsoft SQL Server\MSSQL13.MSSQLSERVER\mssql\binn\rlauncher.config"
```

Edit Configuration Properties

The following table lists each of the settings supported for SQL Server 2017, with the permissible values.

All settings take the form of a key-value pair, with each setting on a separate line. For example, this property specifies the trace level for RLauncher:

Default: TRACE_LEVEL=4

SETTING NAME	VALUE TYPE	DEFAULT	DESCRIPTION
JOB_CLEANUP_ON_EXIT	Integer 0 = Disabled 1 = Enabled	1 Log files are removed on exit	Specifies whether the temporary working folder created for each R session should be cleaned up after the R session is completed. This setting is useful for debugging. Note: This is an internal setting only – do not change this value.
TRACE_LEVEL	Integer 1 = Error 2 = Performance 3 = Warning 4 = Information	1 Output warnings only	Configures the trace verbosity level of the R launcher (MSSQLLAUNCHPAD) for debugging purposes. This setting affects the verbosity of the traces stored in the following trace files, both of which are located in the path specified by the LOG_DIRECTORY setting: **rlauncher.log**: The trace file generated for R sessions launched by T-SQL queries.

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\MSSQLLaunchpad, 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.

See Also

Security Considerations

Security Considerations for the R Runtime in SQL Server

4/19/2017 • 2 min to read • Edit Online

This topic provides an overview of security considerations for working with R Services (In-Database) in SQL Server 2017.

For more information about managing the service, and about how to provision the user accounts used to execute R scripts, see Configure and Manage Advanced Analytics Extensions.

Use Firewall to Restrict Network Access by R

In the suggested installation method, a Windows Firewall rule is used to block all outbound network access from the R runtime processes. Firewall rules should be created to prevent the R runtime process from downloading packages or from making other network calls that could potentially be malicious.

We strongly recommend that you turn on Windows Firewall (or another firewall of your choice) to block network access by the R runtime.

If you are using a different firewall program, you can also create rules to block outbound network connection for the R runtime, by setting rules for the local user accounts or for the group represented by the user account pool. For more information, see Configure and Manage Advanced Analytics Extensions.

Authentication Methods Supported for Remote Compute Contexts

R Services (In-Database) now supports both Windows Integrated Authentication and SQL logins when creating connections between SQL Server and a remote data science client.

For example, if 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, if your Windows account has the necessary permissions, all R code will be executed on the SQL Server computer. Moreover, any SQL queries executed as part of the R code will be run under your credentials as well.

Although a SQL login can also be used in the connection string for a SQL Server data source, use of a login requires that the SQL Server instance allow mixed mode authentication.

Implied authentication

In general the SQL Server Trusted Launchpad starts the R runtime and executes R scripts under its own account. However, if the R script makes an ODBC call, the SQL Server Trusted Launchpad will impersonate 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, **SQLRUser**) must have an account in the master database for the instance, and this account must be given permissions to connect to the instance.

No Support for Encryption at Rest

Transparent Data Encryption is not supported for data sent to or received from the R runtime. As a consequence, encryption at rest **will not** be applied to any data that you use in R scripts, any data saved to disk, or any persisted intermediate results.

See Also

enter link description here

Modify the User Account Pool for SQL Server R Services

4/19/2017 • 5 min to read • Edit Online

As part of the installation process for R 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 R scripts by different SQL users.

This topic describes the default configuration, security and capacity for the worker accounts, and how to change the default configuration.

Worker Accounts Used by R Services

The Windows account group is created by SQL Server setup for each instance on which R Services is installed. Therefore, if you have installed multiple instance that support R, there will be multiple user groups.

- In a default instance, the group name is **SQLRUserGroup**.
- 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 R sessions, but you can change the number of accounts.

- 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**.

How to change the number of R 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. R scripts will fail if 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 queries concurrently in R. The default is 20 accounts.
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 service.

Managing R Workload

The number of accounts in this pool determines how many R sessions can be active simultaneously. By default, 20 accounts are created, meaning that 20 different users can have active R sessions at one time. If you anticipate the need for more concurrent users executing R scripts, you can increase the number of worker accounts.

When the same user executes multiple R scripts concurrently, all the sessions run by that user will use the same worker account. For example, a single user might have 100 different R scripts running concurrently, as long as resources permit, using a single worker account.

The number of worker accounts that you can support, and the number of concurrent R 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.

In R Services, the resources that can be used by 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 R scripts by configuring an external resource pool.

For additional information about managing R script 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 he 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 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
```

See Also

Configuration (SQL Server R Services)

Advanced Configuration Options for Machine Learning Services

4/29/2017 • 4 min to read • Edit Online

This article describes changes that you can make after setup, to modify the configuration of the R 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 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 the R runtime process. You can modify the number of users if needed to support R. Your database administrator must also give this group permission to connect to any instance where R Services has been enabled. For more information, see Modify the User Account Pool for SQL Server R Services.

However, an access control list (ACL) can be defined for sensitive resources on the SQL Server to deny access to this group to prevent the R runtime process from getting access to the resources.

- The user account pool is linked to a specific instance. For each instance on which R script has been enabled, a separate pool of worker accounts are created. Accounts cannot be shared between instances.

- User account names in the pool are of the format SQLInstanceName*nn*. For example, if you are using the default instance as your R server, 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 R runtime sessions that can be launched simultaneously is limited by the size of this user account pool. However, this limit can be changed by an administrator by using 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 R Services.

Manage Memory Used by External Script Processes

By default, the R runtime processes associated with R Services (In-Database) are limited to using no more than 20% of total machine memory. However, this limit can be increased by the administrator, if needed.

Generally, this amount will be inadequate for serious R tasks such as training model or predicting on many rows of data. You might need to reduce the amount of memory reserved for SQL Server (or for other services) and use Resource Governor to define an external resource pool or pools and allocate. For more information, see Resource Governance for R.

Change Advanced Service Options using the Configuration File

You can control some advanced properties of R Services (In-Database) by editing the R Services (In-Database) configuration file. This 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.

For example, to use Notepad to open the configuration file for the default instance, you would open a command prompt as administrator, and type the following command:

```
C:\>Notepad.exe "%programfiles%\Microsoft SQL Server\MSSQL13.MSSQLSERVER\mssql\binn\rlauncher.config"
```

Edit Configuration Properties

The following table lists each of the settings supported for SQL Server 2017, with the permissible values.

All settings take the form of a key-value pair, with each setting on a separate line. For example, this property specifies the trace level for RLauncher:

Default: TRACE_LEVEL=4

SETTING NAME	VALUE TYPE	DEFAULT	DESCRIPTION
JOB_CLEANUP_ON_EXIT	Integer 0 = Disabled 1 = Enabled	1 Log files are removed on exit	Specifies whether the temporary working folder created for each R session should be cleaned up after the R session is completed. This setting is useful for debugging. Note: This is an internal setting only – do not change this value.
TRACE_LEVEL	Integer 1 = Error 2 = Performance 3 = Warning 4 = Information	1 Output warnings only	Configures the trace verbosity level of the R launcher (MSSQLLAUNCHPAD) for debugging purposes. This setting affects the verbosity of the traces stored in the following trace files, both of which are located in the path specified by the LOG_DIRECTORY setting: **rlauncher.log**: The trace file generated for R sessions launched by T-SQL queries.

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\MSSQLLaunchpad, 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.

See Also

Security Considerations

Deploy and Consume Analytics

4/29/2017 • 3 min to read • Edit Online

Microsoft R Server includes an operationalization feature that make it easier to:

- Publish and manage R and Python models and code in the form of web services
- Consume these services within client applications to affect business results

SQL Server 2017 CP 2.0 now includes this feature as an option, although it was not installed in previous versions of SQL Server R Services. This topic provides information about how to enable and configure the feature. This topic provides information about how to enable and configure the feature.

What's New in This Feature

- Role-based access control to analytical web services. These roles 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. Learn more about roles.

- Scoring performance

  Use real time scoring of web services with a supported R model object to improve the speeed of scoring operations

- Publish Python code as a web service

  For more information, see Publish and consume Python code.

- Asynchronous batch consumption for large input data

  Web services can now be consumed asynchronously via batch execution.

- Autoscaling of a grid of web and compute nodes on Azure

  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 the 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.

Background

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 Machine Learning Server provides support for deployment through web services that support publishing of R jobs, and an administrative utility for running distributed R jobs.

Typically, you would not install Machine Learning Server on the same computer that is running SQL Server Machine Learning Services. It is possible, but we recommend that you keep them separate. In other words, install **Microsoft Machine Learning Server** on a separate computer form SQL Server, and then configure the operationalization features on that computer.

For general information about scenarios supported by operationalization, see Operationalization with R Server.

Deploying and consuming web services For the Data Scientist For the Application Developer For the Administrator

Requirements

The **mrsdeploy** package is installed when you use the option to install **Microsoft Machine Learning Server**, from the **Shared Features** section of SQL Server setup.

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. For more information, see mrsdeploy functions.

To actually use the feature, some additional steps are required.

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**, you must manually add a registry key for use by **mrsdeploy**, that specifies the base folder for the R_SERVER files. In a default installation, that path is as follows: `C:\Program Files\Microsoft SQL Server\140\R_SERVER`
   a. Create a new registry key as follows: `H_KEY_LOCAL_MACHINE\SOFTWARE\R Server\Path`
   b. Set the value of the key to `"C:\Program Files\Microsoft SQL Server\140\R_SERVER"`.
4. When done, open the Administrator Utility. For help, see R Server help
5. Configure the service as described here: Configuring R Server for Operationalization

Managing and Monitoring R Solutions

4/19/2017 • 1 min to read • Edit Online

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. R Services (In-Database) provides many benefits to the database administrator who supports the data science role.

- **Security.** 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.

- **Reliability.** R sessions are executed in a separate process to ensure that your server continues to run as usual even if the R session encounters issues. Low privilege physical user accounts are used to contain and isolate R instances.

- **Resource governance.** You can control the amount of resources allocated to the R runtime, to prevent massive computations from jeopardizing the overall server performance.

In This Section

Monitoring R Services

Resource Governance for R Services

Installing and Managing R Packages

Configuration

- Configure and Manage Advanced Analytics Extensions

- Modify the User Account Pool for SQL Server R Services

  Security Considerations for the R Runtime in SQL Server

See Also

SQL Server R Services Features and Tasks

Resource Governance for R Services

4/19/2017 • 3 min to read • Edit Online

One pain point with R is that analyzing large amounts of data in production requires additional hardware, and data is often moved outside the database to computers not controlled by IT. To perform advanced analytics operations, customers want to leverage database server resources, and to protect their data, they require that such operations meet enterprise-level compliance requirements, such as security and performance.

This section provides information about how you can manage resources used by the R runtime and by R jobs running using the SQL Server instance as the compute context.

What is Resource Governance?

Resource governance is designed to identify and prevent problems that are common in a database server environment, where there are often multiple dependent applications and multiple services to support and balance. For R Services (In-Database), 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 R jobs are running concurrently on the server and the jobs are not isolated from each other by using resource pools, the resulting resource contention could lead to unpredictable performance or threaten completion of the workload.

- Prioritizing workloads.

  The administrator or architect needs to be able to specify workloads that must take precedence, or guarantee certain workloads to complete if there is resource contention.

  In R Services (In-Database), you can use Resource Governor to manage the resources used by the R runtime and by remote R jobs.

How to Use Resource Governor to Manage R jobs

In general, you manage resources allocated to R jobs by creating *external resource pools* and assigning workloads to the pool or pools. An external resource pool is a new type of resource pool introduced in SQL Server 2017, to help manage the R runtime and other processes external to the database engine.

In SQL Server 2017, there are now 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 using 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: How To: Create a Resource Pool for R

You can use the *external resource pool* mechanism to manage the resources used by the following R executables:

- Rterm.exe and satellite processes
- BxlServer.exe and satellite processes
- Satellite processes launched by LaunchPad

  However, 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 also 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 R Services (In-Database), 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 R Solutions
How To: Create a Resource Pool for R
Resource Governor Resource Pool

How To: Create a Resource Pool for R

4/19/2017 • 4 min to read • Edit Online

This topic describes how you can create a resource pool specifically for managing R workloads in R Services (In-Database). It assumes that you have already installed R Services (In-Database) and want to reconfigure the SQL Server 2017 instance to support more fine-grained management of the resources used by R.

For more information about managing server resources, see Resource Governor and Resource Governor Related Dynamic Management Views (Transact-SQL).

Steps

1. Review status of existing resource pools
2. Modify server resource pools
3. Create a new resource pool for external processes
4. Create a classification function to identify R requests
5. Verify that new external resource pool is capturing R jobs

Review the status of existing resource pools

1. First, check the resources allocated to the default pool for the server.

```
SELECT * FROM sys.resource_governor_resource_pools WHERE name = 'default'
```

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'
```

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 R 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 SQL Server
   - 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 R requirements against 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.

   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);
   ```

   Note the new **EXTERNAL** keyword.

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 R

1. 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.

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, you 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

1. To verify that the changes have been made, check the configuration of server memory and CPU for each of the workload groups associated with all instance resource pools: the default pool for the SQL Server server, the default resource pool for external processes, and the user-defined pool for external processes.

```
SELECT * FROM sys.resource_governor_workload_groups;
```

Results

GROUP_ID	NAME	IMPORTANCE	REQUEST_MAX_MEMORY_GRANT_PERCENT	REQUEST_MAX_CPU_TIME_SEC	REQUEST_MEMORY_GRANT_TIMEOUT_SEC	MAX_DOP	GROUP_MAX_REQUESTS POOL_ID	POOL_IDD	EXTERNAL_POOL_IDD
1	internal	Medium	25	0	0	0	0	1	2

GROUP_ID	NAME	IMPORTANCE	REQUEST_MAX_MEMORY_GRANT_PERCENT	REQUEST_MAX_CPU_TIME_SEC	REQUEST_MEMORY_GRANT_TIMEOUT_SEC	MAX_DOP	GROUP_MAX_REQUESTS POOL_ID	POOL_IDD	EXTERNAL_POOL_IDD
2	default	Medium	25	0	0	0	0	2	2
256	ds_wg	Medium	25	0	0	0	0	2	256

2. You can 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;
```

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. The following statement returns information about the computer resources that are affinitized to the external resource pool.

```
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

Resource Governor
Resource Governance for R Services

Extended Events for SQL Server R Services

4/19/2017 • 5 min to read • Edit Online

R Services (In-Database) provides a set of extended events to use in troubleshooting operations related to the SQL Server Trusted Launchpad or R jobs sent to SQL Server.

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';
```

However, some additional extended events for R Services (In-Database) are fired only from external processes, such as the SQL Server Trusted Launchpad, and BXLServer, the satellite process that starts the R runtime. For more information about how to capture these events, see Collecting Events from External Processes.

For general information about using extended events, see SQL Server Extended Events Sessions.

Table of Extended Events

EVENT	DESCRIPTION	USE
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	
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.

EVENT	DESCRIPTION	USE
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 (just before 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	
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.

EVENT	DESCRIPTION	USE
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

R Services (In-Database) 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 R and other external script languages.

  To capture events related to BXLServer, place the *.config* file in the R installation directory. In a default installation, this would be:

  ```
  C:\Program Files\Microsoft SQL
  Server\MSSQL_version_number.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64
  ```

  .

IMPORTANT

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>
```

Notes:

- 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 (such as `[XEvent Name 1]`, `[XEvent Name 1]`).

- 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>
```

Notes:

- 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>
```

Notes:

- 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 R Services
SQL Server R Services
Managing and Monitoring R Solutions

DMVs for SQL Server R Services

4/19/2017 • 6 min to read • Edit Online

The topic lists the system catalog views and DMVs that are related to R Services (In-Database).

For information about extended events, see Extended Events for SQL Server R Services.

> **TIP**
>
> The product team has provided custom reports that you can use to monitor R Services sessions and packages. For more information, see Monitor R Services using Custom Reports in Management Studio.

System Configuration and System Resources

You can monitor and analyze the resources used by R scripts by using SQL Server system catalog views and DMVs.

General

- 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 '%Extended Scripts%'
  ```

  The following counters are reported by this DMV for external scripts per instance:

  - **Total Executions**: Number of R 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 R scripts run where the call was instantiated remotely and SQL Server 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 R calls are made on an instance. The telemetry service starts when SQL Server does and increments a disk-based counter each time a specific R function is called.

  The counter is incremented per call to a 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

- 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 R jobs are run in 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_os_schedulers (Transact-SQL).

> **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 R jobs.

For general information about monitoring SQL Server instances, see Catalog Views and Resource Governor Related Dynamic Management Views.

R Script Execution and Monitoring

R 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 R 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 R 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.

See Also

Managing and Monitoring

Using R Code Profiling Functions

4/19/2017 • 1 min to read • Edit Online

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 R Services using Custom Reports in Management Studio

4/19/2017 • 3 min to read • Edit Online

To make it easier to manage SQL Server R Services, the product team has provided a number of sample custom reports that you can add to SQL Server Management Studio, to view R Services details such as:

- A list of the active R sessions
- The R configuration of the current instance
- Execution statistics for the R runtime
- A list of extended events for R Services
- A list of R packages installed on the current instance

This topic explains how to install and use the reports. For more information about custom 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 from this page:

   - SSMS Custom Reports

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 for SQL Server R Services:

- **R Services - Active Sessions**

  Use this report to view the users who are currently connected to the SQL instance and running R jobs.

- **R Services - Configuration**

  Use this report to view the properties of the R runtime and configuration of R Services. The report will indicate whether a restart is required, and will check for required network protocols.

  Implied authentication is required for running R in a SQL compute context, To check this, the report verifies whether a database login exists for the group SQLRUserGroup.

  > **NOTE**
  >
  > For more information about these fields, see Package metadata, by Hadley Wickam. For example, the *Nickname* field for the R runtime was introduced to help differentiate between releases.

  - **R Services - Configure Instance**

    This report is intended to help you configure R Services after installation. You can run it from the preceding report if R Services is not configured correctly.

- **R Services - Execution Statistics**

  Use this report to view the execution statistics of R Services. 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. Currently the report monitors only statistics for RevoScaleR package functions. Click **View SQL Script** to get the T-SQL code for this report.

- **R Services - Extended Events**

  Use this report to view a list of the extended events that are available for monitoring R script execution. Click **View SQL Script** to get the T-SQL code for this report.

- **R Services - Packages**

  Use this report to view a list of the R packages installed on the SQL Server instance. Currently the report includes these package properties:

  - Package (name)
  - Version
  - Depends
  - License
  - Built
  - Lib Path

- **R Services - Resource Usage**

  Use this report to view consumption of CPU, memory, and I/O resources by SQL Server R scripts execution.

You can also view the memory setting of external resource pools.

See Also

Monitoring R Services

Extended events for R Services

Known Issues for Machine Learning Services

4/29/2017 • 18 min to read • Edit Online

This topic describes limitations and known issues with the following machine learning components that are provided as an option in SQL Server 2016 and SQL Server 2017.

- 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

Additional guidance related to initial setup and configuration are listed here: Upgrade and Installation FAQ.

The articles contains details on how to upgrade from previous versions, including earlier versions of the Revolution Analytics tools and libraries, side-by-side installation, and solutions for some common problems related to installing and uninstalling the R features.

License agreement for machine learning components required for unattended installs

If you use the command line to install an instance of SQL Server 2016 or SQL Server 2017, and add the feature that supports use of external languages such as R or Python, you must add a separate licensing agreement parameter to the command line arguments for each language that you install.

- For R: */IACCEPTROPENLICENSEAGREEMENT*
- For Python: */IACCEPTPYTHONLICENSEAGREEMENT*

Failure to use the correct argument will cause SQL Server setup to fail.

Install 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 using 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 the Microsoft R Server version 8.x.x. Download and install a compatible version.

SQL Server 2016 required that the R libraries on the client exactly match the R libraries on the server. That 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 used by your client and the server, nad 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. Therefore, to ensure that you always have the most up-to-date versions of R components, you should install all service packs.

For compatibility with Microsoft R Client 9.0.0, you must 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 to the Modern Lifecycle policy as described in this section. When you do so, the version of R installed with SQL Server is updated on the same schedule that updates are published for Microsoft R Server, ensuring that both server and client can always have the latest releases of Microsoft R.

Warning of incompatible version when connecting to older version of SQL Server R Services from a client using SQL Server 2017

If you installed Microsoft R Server on a client computer using the setup wizard for SQL Server 2017 or the new standalone installer for Microsoft R Server, and run R code in a compute context that uses an earlier version of SQL Server R Services, you might see an error like the following:

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.

The **SqlBindR.exe** tool is provided in the Microsoft R Server 9.0 release to support upgrade of SQL Server instances to a compatible 9.0 version. Support for upgrade of R Services instances to 9.0 will be added in SQL Server as part of an upcoming service release. Versions that are candidates for future upgrade include SQL Server 2016 RTM CU3+ and SP1+, and SQL Server 2017 CTP 1.1.

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 typically occurs when multiple components are 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 /IACCEPTROPENLICENSETERMS
/MRCACHEDIRECTORY=<path to CU1 CAB files>
```

To get the latest installers, see Installing Machine Learning Components without Internet Access.

SQLRUserGroup for Launchpad must have an account in the SQL Server instance

When Machine Learning Services is installed, setup creates a new Windows user group, with the default name **SQLRUserGroup**, which is used by Launchpad to run R jobs. If you need to run R or Python jobs from a remote client using Windows integrated authentication, you must give this Windows user group permission to log into the SQL Server instance where R is enabled.

In an environment where the group **SQLRUserGroup** does not have this permission, you might see the following errors:

- When trying 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\MSSQLLAUNCHPAD was unable to log in.

> **NOTE**
>
> If you run R jobs in SQL Server Management Studio using Shared Memory, you might not encounter this limitation until your R job uses an embedded ODBC call.
>
> This workaround is not required if you use SQL logins from the remote workstation.

Launchpad services fails to start if version is different than R version

If you install R Services separately from the database engine, and the build versions are different, you might see this 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 using the release version, apply a patch to upgrade the database engine, and then add R Services using the release version.

To avoid this problem, make sure that all components have the same version number. If you upgrade one component, be sure to apply the same upgrade to all other installed components.

To view a list of the R version numbers required for each release of SQL Server 2016, see Installing R components without Internet Access.

Service account for LaunchPad requires permission Replace Process Level Token

On installation of SQL Server R Services, the Trusted Launchpad is started using the account NT Service\MSSQLLaunchpad, which by default is provisioned with necessary permissions. However, if you use a different account, or change the privileges associated with this account, the Launchpad might fail to start.

To ensure that the Launchpad service account can log in, give the account the privilege: `Replace Process Level Token`. For more information, see Replace a process level token.

Remote compute contexts blocked by firewall in SQL Server instances 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 use of the virtual machine's workspace. The reason is that, by default, the Azure VM firewall 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”.

Implied authentication in SQLEXPRESS

When you run R jobs from a remote data science workstation using Windows integrated authentication, SQL Server will use *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 you cannot upgrade, you can use a SQL login to run remote R jobs that might require embedded ODBC calls.

Multiple R libraries and executables are installed if you install both Standalone and In-Database features

SQL Server setup includes the option to install Microsoft R Server (Standalone). The Microsoft R Server (Standalone) option can be used in Enterprise Edition to install a standalone Windows server that supports R but that does not require interactivity with SQL Server.

However, this standalone option is **not** required in order to use R with Transact-SQL.

Moreover, we recommend that you **do not** install both the In-database and Standalone features on the SQL Server computer. If you need to set up client tools capable of connecting to R In-Database, we recommend Microsoft R Client.

If you install both features (In-Database and Standalone) on the SQL Server computer, be aware that separate copies of the R libraries and R tools will be created for each instance of SQL Server that uses R, and another copy of the R libraries and tools for the Standalone install. Therefore, if you installed R for both the default instance and a named instance of SQL Server, and the R Server (Standalone), you might have three instances of R on the same computer:

- **SQL Server 2016**
  - R Services, default instance: `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES`
  - R Services, named instance: `C:\Program Files\Microsoft SQL Server\MSSQL13.<instance_name>\R_SERVICES`
  - R Server Standalone: `C:\Program Files\Microsoft SQL Server\130\R_SERVER`
- **SQL Server 2017:**
  - Machine Learning Services, default instance: `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES`
  - Machine Learning Services, named instance: `C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\R_SERVICES`
  - Machine Learning Server Standalone: `C:\Program Files\Microsoft SQL Server\140\R_SERVER`

> **NOTE**
>
> If you install R Server (Standalone), by default the R libraries and tools are installed by default in `C:\Program Files\Microsoft SQL Server\140\R_SERVER`. However, if you install R Server using the separate Windows installer, the libraries are installed in `C:\Program Files\Microsoft\R Server\R_SERVER`. For more information, see Install R Server 9.0.1 for Windows.

Having multiple copies of the R tools and libraries can lead to confusion when you are trying to install new R packages or test your code. If you are developing solution that are meant to be run in SQL Server, we recommend the following:

- Use the R libraries and tools associated with the database instance only from the context of Transact-SQL, to avoid resource contention.
- Work from a remote client and install your R development environment and the Microsoft R libraries on that computer. Connect to SQL Server to run your code using a SQL Server compute context.

Performance limits when R libraries are called from standalone R tools

It is possible to call the R tools and libraries that are installed for SQL Server R Services from an external R application such as RGui. This might be handy when you are installing new packages, or running ad hoc tests on very short code samples.

However, be aware that outside of SQL Server, performance will be limited. For example, even if you have purchased the Enterprise Edition of SQL Server, R will run in single-threaded mode when you run your R code using external tools. Performance will be superior if you run your R code by initiating a SQL Server connection and using sp_execute_external_script, which will call the R libraries for you.

- Avoid calling the R libraries used by SQL Server from external R tools.
- If you need to run extensive R code on the SQL Server computer without using SQL server, install a separate instance of R such as Microsoft R Client, and then ensure that your R development tools point to the new library.

For more information, see Create a Standalone R Server.

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%. Therefore, if the server had 32GB of RAM, the R executables (RTerm.exe and BxlServer.exe) could use a maximum 6.4GB in a single request.

If you encounter resource limitations, check the current default, and if 20% is not enough, see the documentation for SQL Server on how to change this value.

R Code Execution and Package or Function Issues

This section contains known issues that are specific to running R on SQL Server, as well as some issues related to the R libraries and tools published by Microsoft, including RevoScalerR.

For additional known issues that might affect R solutions, see the Microsoft R Server site: Known Issues with Microsoft R Server

Limitations on processor affinity for R jobs

In the RTM 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 2 k-groups, only processors from the first k-group will be used for the R processes. The same limitation applies when configuring resource governance for R script jobs.

This issue is fixed in SQL Server 2016 Service Pack 1.

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"))
```

This issue will be fixed in a later release.

As a workaround, you can rewrite the SQL query to use CAST or CONVERT and present the data to R using the correct data type. In general, it is better for performance to work with data using SQL rather than changing data in the R code.

Avoid clearing workspaces when executing R code in a SQL Server compute context

If you use the 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 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: `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.

Arguments *varsToKeep* and *varsToDrop* not supported for SQL Server data sources

When you use the rxDataStep function to write results to a table, using the *varsToKeep* and *varsToDrop* is a handy way of specifying the columns to include or exclude as part of the operation. Currently, these arguments are not supported for SQL Server data sources.

This limitation will be removed in a later release.

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 Working with R 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 different collations and languages that are supported in R and Transact-SQL. Any string in a non-ASCII encoding can potentially be handled incorrectly.

When sending string data to R, convert it to an ASCII representation, if possible.

Only one raw value 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 the value in the output data frame.

Support for multiple **raw** outputs will be added in subsequent releases.

One possible workaround if multiple output sets are desired is to do multiple calls of the stored procedure and send the result sets back to SQL Server using ODBC.

Note that you can return parameter values together with the results of the stored procedure simply by adding the OUTPUT keyword. For more information, see Returning Data by Using OUTPUT Parameters.

Loss of precision

Transact-SQL and R support different data types; therefore, numeric data types can suffer loss of precision during conversion.

For more information about implicit data type conversion, see Working with R Data Types.

Variable scoping error "The sample data set for the analysis has no variables" when using the transformFunc parameter

You can pass a *transformFunc* argument in a function such as `rxLinmod` or `rxLogit` to transform the data while modelling. 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.

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 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 as follows:

```
g <- function(y){
              f <- function(x) { 2*x +3}
              a <- 10 * y
              f(a)
}
```

Data import and manipulation using RevoScaleR

When reading **varchar** columns from a database, white space will be trimmed. To prevent this, enclose strings in non-white-space characters.

When using functions such as `rxDataStep` to create database tables with **varchar** columns, the column width is estimated based on a sample of the data. If the width can vary, it may 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 provided by the RevoScaleR package can be used only in single-threaded mode.

Parallelism for `rxExec` across multiple processes will be added in an upcoming release.

Increase maximum parameter size to support rxGetVarInfo

If you use data sets with extremely large numbers of variables (e.g., over 40,000), you should set the `max-ppsize` flag when starting R in order 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, in rgui.exe or rterm.exe), you can set the value of max-ppsize to 500000 by typing:

```
R --max-ppsize=500000
```

If you are using the DevelopR environment, you can set the `max-ppsize` flag by making this call to the RevoIDE executable:

```
RevoIDE.exe /RCommandLine --max-ppsize=500000
```

Issues with the rxDTree function

The `rxDTree` 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 will be automatically binned.

Ordered factors are treated the same as factors in all RevoScaleR analysis functions except `rxDTree`.

Revolution R Enterprise and Microsoft R Open

This section lists issues specific to R connectivity, development, and performance tools 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.

Running side by side versions of Revolution R Enterprise

Installing Revolution R Enterprise with side by side with any version of R Services (In-Database) is not supported.

If you have a license to use a different version of 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.

Use of R Productivity Environment Not Supported

Some prerelease versions of R Services (In-Database) included an R development environment for Windows that was created by Revolution Analytics. This tool is not longer provided and is not supported.

For compatibility with R Services (In-Database), we strongly recommend that you install Microsoft R Client or Microsoft R Server instead of the Revolution Analytics tools. R Tools for Visual Studio is another recommended client that 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 and FAQ

4/29/2017 • 1 min to read • Edit Online

NOTE

SQL Server 2017 has brought lots of changes! We are in the process of reorganizing content related to setup and configuration, as well as troubleshooting.

This section will include links to troubleshooting resources such as these:

- How to resolve problems with Launchpad
- Reasons why Python or R script won't run
- Common errors related to external scripts
- Upgrading from older versions or pre-release versions

In the meantime, please see these topics:

Known issues

- Setup FAQ
- Known Issues

Step-by-step setup and configuration

- Set up R Services or Machine Learning Services with R
- Set up Machine Learning Services with Python
- Use SqlBindR to Upgrade an Instance of R services

Perform an offline install or automated install

- Unattended Installation of R Services
- Unattended Installation of Machine Learning Services with Python
- Installing Machine Learning Components without Internet Access

Modify your configuration

- Modify the User Account Pool for SQL Server R Services
- Configure and Manage Advanced Analytics Extensions

Related tools

- Set up Microsoft Machine Learning Server Standalone
- Set up R Server on an Azure VM
- Install R Server for Windows
- Get R Tools for Visual Studio

Upgrade and Installation FAQ (SQL Server R Services)

4/29/2017 • 14 min to read • Edit Online

This topic provides answers to common questions about installation and upgrades. It also contains questions about upgrades from preview releases of R Services (In-Database).

If you are installing R Services (In-Database) for the first time, follow the procedures for setting up SQL Server 2017 and the R components as described here: Set up SQL Server R Services (In-Database).

This topic also contains some issues related to setup and upgrade of R Server (Standalone). For issues related to Microsoft R Server on other platforms, see Microsoft R Server Release Notes and Run Microsoft R Server for Windows.

Important changes from pre-release versions

If you installed any pre-release version of SQL Server 2016, or if you are using setup instructions that were published prior to the public release of SQL Server 2016, it is important to be aware that the setup process is completely different between pre-release and RTM versions. These changes include both the options available in the SQL Server setup wizard and post-installation steps.

> **WARNING**
>
> New installation of any pre-release version of R Services (In-Database) is no longer supported. We recommend that you upgrade as soon as possible.
>
> - If you installed R Services (In-Database) in CTP3, CTP3.1, CTP3.2, RC0, or RC1, you will need to re-run the post-configuration installation script to uninstall the previous versions of the R components and of R Services.
> - The post-configuration installation script is provided solely to help customers uninstall pre-release versions. Do not run the script when installing any newer version.

Upgrading R components

As hotfixes or improvements to SQL Server 2016 are released, R components will be upgraded or refreshed as well, if your instance already includes the R Services feature.

If you install or upgrade servers that are not connected to the Internet, you must download an updated version of the R components manually before beginning the refresh. For more information, see Installing R Components without Internet Access.

If you are using SQL Server 2017, upgrades to R components are automatically installed.

As of December 2016, it is now possible to upgrade R components on a faster cadence than the SQL Server release cycle, by *binding* an instance of R Services to the Modern Software Lifecycle policy. For more information, see Use SqlBindR to Upgrade an Instance of SQL Server R Services

Support for slipstream upgrades

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.

- In SQL Server 2016, you can start a slipstream upgrade in SQL Server Management Studio by clicking **Tools**, and selecting **Check for Updates**.
- If the server does not have Internet access, be sure to download the SQL Server installer. You must also separately download matching versions of the R component installers **before** beginning the update process. For download locations, see Installing R Components without Internet Acccess.

  When all setup files have been copied to a local directory, start the setup utility by typing SETUP.EXE from the command line.

  - Use the */UPDATESOURCE* argument to specify the location of a local file containing the SQL Server update, such as a Cumulative Update or Service Pack release.
  - Use the */MRCACHEDIRECTORY* argument to specify the folder containing the R component CAB files.

For more information, see this blog by the R Services Support team: Deploying R Services on Computers without Internet Access.

New license agreement for R components required for unattended installs

If you use the command line to upgrade an instance of SQL Server 2017 that already has R Services (In-Database) installed, you must modify the command line to use the new license agreement parameter, */IACCEPTROPENLICENSEAGREEMENT*.

Failure to use the correct argument can cause SQL Server setup to fail.

After running setup, R Services (In-Database) is still not enabled

The feature that supports running external scripts is disabled by default, even if installed. This is by design, for surface area reduction.

To enable R scripts, an administrator can run the following statement in SQL Server Management Studio:

1. On the SQL Server 2017 instance where you want to use R, run this statement.

   ```
   sp_configure 'external scripts enabled',1
   reconfigure with override
   ```

2. Restart the instance.
3. After the instance has restarted, open **SQL Server Configuration Manager** or the **Services** panel and verify that the SQL Server Trusted Launchpad service is running.

> **TIP**
>
> To install an instance of R Services that is preconfigured, use the Azure Virtual machine image that includes Enterprises Edition with R Services enabled. For more information, see Installing SQL Server R Services on an Azure Virtual Machine.

Setup of R Services (In-Database) not available in a failover cluster

Currently, it is not possible to install R Services (In-Database) on a failover cluster.

However, you can install R Services (In-Database) on a standalone computer that uses Always On and is part of an availability group. For more information about using R Services in an Always On availability group, see Always On Availability Groups: Interoperability.

Another option is to set up a replica on a different SQL Server instance for the purpose of running R scripts. The replica could be created using either replication or log shipping.

Launchpad service cannot be started

There are several issues that can prevent Launchpad from starting.

8dot3 notation is not enabled.

To install R Services (In-Database), the drive where the feature is installed must support creation of short file names using the **8dot3** notation. An 8.3 filename is also called a short filename and is used for compatibility with older versions of Microsoft Windows prior or as an alternate filename to the long filename.

If the volume where you are installing R Services (In-Database) does not support the short filenames, the processes that launch R from SQL Server might not be able to locate the correct executable and the SQL Server Trusted Launchpad will not start.

As a workaround, you should enable the 8dot3 notation on the volume where SQL Server is installed and 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 file RLauncher.config. In a default installation, the file RLauncher.config is located in C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Binn
3. Make a note of the property for WORKING_DIRECTORY.
4. Use the fsutil utility with the *file* argument to specify a short file path for the folder specified in WORKING_DIRECTORY.
5. Edit the configuration file to use the short name for WORKING_DIRECTORY.

Alternatively, you can specify a different directory for WORKING_DIRECTORY that has a path compatible with the 8dot3 notation.

> **NOTE**
>
> This restriction has been removed in later releases. If you experience this issue, please install one of the following:
>
> - SQL Server 2016 SP1 and CU1: Cumulative Update 1 for SQL Server
> - SQL Server 2016 RTM, Service Pack 3, and this hotfix, available on demand

The account that runs Launchpad has been changed or necessary permissions have been removed.

By default, SQL Server Trusted Launchpad uses the following account on startup, which is configured by SQL Server setup to have all necessary permissions: `NT Service\MSSQLLaunchpad`

Therefore, if you assign a different account to the Launchpad or the right is removed by a policy on the SQL Server machine, the account might not have 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 give the necessary permissions to the new service account, use the **Local Security Policy** application and update the permissions on the 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)

User group for Launchpad is missing system right "Allow log in locally"

During setup of R Services, SQL Server creates the Windows user group, **SQLRUserGroup**, and by default provisions it with all rights necesssary for Launchpad to connect to SQL Server and run external script jobs.

However, in organizations where more restrictive security policies are enforced, this right might be manually removed, or revoked by policy. If this happens, Launchpad can no longer connect to SQL Server, and R Services will be unable to function.

To correct the problem, ensure that the group **SQLRUserGroup** has the system right **Allow log on locally**.

For more information, see Configure Windows Service Accounts and Permissions.

Error: Unable to communicate with the Launchpad service

If you install R Services and enable the feature, but get this error when you try to run an R script, it might be that the Launchpad service for the instance has 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 where R Services is not working, and select **Properties**.
3. Click the **Service** tab and verify that the service is running. If it is not, change the **Start Mode** to **Automatic** and click **Apply**.
4. Typically, restarting the service enables R scripts. If it does not, make a note of the path and arguments in the **Binary Path** property.
    - Review the rlauncher.config file and ensure that the working directory is valid
    - Ensure that the Windows group used by Launchpad has the ability to connect to the SQL Server instance, as described in the previous section.
    - Restart the Launchpad service if you make any changes to service properties.

Side by side installation not supported

Do not create a side-by-side installation using another version of R or other releases from Revolution Analytics.

Offline installation of R components for localized version of SQL Server

If you are installing R Services on a computer that does not have Internet access, you must take two additional steps: you must download the R component installer to a local folder before you run SQL Server setup, and you must edit the installer file to ensure that the correct language is installed.

The language identifier used for the R components must be the same as the SQL Server setup language being installed, or the **Next** button is disabled and you cannot complete setup.

For more information, see Installing R Components without Internet Access.

Unable to launch runtime for R script

R Services (In-Database) creates a Windows users group that is used by the SQL Server Trusted Launchpad to run R jobs. This user group must have the ability to log into the instance that is running R Services in order to execute R on the behalf of remote users who are using Windows integrated authentication.

In an environment where the Windows group for R users (**SQLRUsers**) does not have this permission, you might see the following errors:

- When trying 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\MSSQLLAUNCHPAD was unable to log in.

For information about how to give this user group the necessary permissions, see Set up SQL Server R Services.

> **NOTE**
>
> This limitation does not apply if you use SQL logins to run R scripts from a remote workstation.

Remote execution via ODBC

If you use a data science workstation and connect to the SQL Server computer to run R commands using the **RevoScaleR** functions, you might get an error when using ODBC calls that write data to the server.

The reason is the same as described in the previous section: at setup, R Services creates a group of worker accounts that are used for running R tasks. However, if these accounts cannot connect to the server, ODBC calls cannot be executed on your behalf.

Note that 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.

To enable implied authentication, you must give this group of worker accounts permissions as follows:

1. Open SQL Server Management Studio as an administrator on the instance where you will 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 name.

   ```
   USE [master]
   GO

   CREATE LOGIN [computername\SQLRUserGroup] FROM WINDOWS WITH DEFAULT_DATABASE=[master],
   DEFAULT_LANGUAGE=[language]
   GO
   ```

For more information and the steps for doing this using the SQL Server Management Studio UI, see Set up SQL Server R Services.

How to uninstall previous versions of R Services

It is important that you uninstall previous versions of R Services (In-Database) and its related R components in the correct order, particularly if you installed any of the pre-reelease versions.

Step 1. Run script to deregister Windows user group and components before uninstalling previous components

If you installed a pre-release version of R Services (In-Database), you must first run run the script `RegisteREext.exe` with the `/uninstall` argument.

By doing so, you deregister old components and remove the Windows user group associated with Launchpad. If you do not do this, you will be unable to create the Windows user group required for any new instances that you install.

For example, if you installed R Services on the default instance, run this command from the directory where the script is installed:

```
    RegisterRExt.exe /UNINSTALL
```

If you installed R Services on a named instance, specify the instance name after *INSTANCE:*

```
RegisterRExt.exe /UNINSTALL /INSTANCE:<instancename>
```

You might need to run the script more than once to remove all components.

Important: The default location for this script is different, depending on the pre-release version you installed. If you try to run the wrong version of the script, you might get an error.

- **CTP 3.1, 3.2, or 3.3**

  Additional steps are required to uninstall existing components.

  1. First, download an updated version of the post-installation configuration script from the Microsoft Download Center. The updated script supports de-registration of older components.
  2. Click the link and select **Save As** to save the script to a local folder.
  3. Rename the existing script, and then copy the new script into the folder where the script will be executed.

- **RC0**

  The script file is located in this folder: `C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\rxLibs\x64`

- **Release versions (13.0.1601.5 or later)**

  No script is needed to install or configure components. The script should be used solely for removing older components.

Step 2. Uninstall any older versions of the Revolution Enterprise tools, including components installed with CTP releases.

The order of uninstallation of the R components is critical. Always uninstall Microsoft R Enterprise first. Then, uninstall Microsoft R Open.

If you mistakenly uninstall R Open first and then get an error when trying to uninstall Revolution R Enterprise, one workaround is to reinstall Revolution R Open or Microsoft R Open, and then uninstall both components in the correct order.

Step 3. Uninstall any other version of Microsoft R Open.

Finally, uninstall all other versions of Microsoft R Open.

Step 4. Upgrade SQL Server

After all pre-release components have been uninstalled, restart the computer. This is a requirement of SQL Server setup and you will not be able to proceed with an updated installation until a restart is completed.

After this is done, run SQL Server setup and follow these instructions to install R Services (In-Database): Set up SQL Server R Services.

No additional components are needed; the R packages and connectivity packages are installed by SQL Server setup.

Problems uninstalling older versions of R

In some cases, older versions of Revolution R Open or Revolution R Enterprise are not completely removed by the uninstall process.

If you have problems removing an older version, you can also edit the registry to remove related keys.

Open the Windows Registry, and locate this key: `HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall`.

Delete any of the following entries, if present, and if the key contains only a single 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)

Upgrade of R Server (Standalone) to RC3 requires uninstallation using the RC2 setup utility

Microsoft R Server (Standalone) first became available in SQL Server 2017 RC2. To upgrade to the RC3 version of Microsoft R Server, you must first uninstall using SQL Server 2017 RC2 setup, and then reinstall using SQL Server 2017 RC3 setup.

1. Uninstall R Server (Standalone) for SQL Server 2017 RC2 using SQL Server setup.
2. Upgrade SQL Server 2017 to RC3 and select the option to install R Services (In-Database). This upgrades the instance of R Services (In-Database) to RC3; no additional configuration is necessary.
3. Run SQL Server 2017 setup for RC3 once more, and install Microsoft R Server (Standalone).

> **NOTE**
>
> This workaround is not required when upgrading to the RTM version of Microsoft R Server.
>
> For additional issues related to Microsoft R Server, see Microoft R Server Release Notes.

See Also

Getting Started with SQL Server R Services
Getting Started with Microsoft R Server (Standalone)

Installing SQL Server R Services on an Azure Virtual Machine

4/19/2017 • 3 min to read • Edit Online

If you deploy an Azure virtual machine that includes SQL Server 2017, you can now select R Services as a feature to be added to the instance when the VM is created.

- Create a new VM with SQL Server 2016 and R Services
- Add R Services to an existing virtual machine with SQL Server 2016

Create a new SQL Server 2016 Enterprise Virtual Machine with R Services Enabled

1. In the Azure Portal, click VIRTUAL MACHINES and then click NEW.
2. Select SQL Server 2016 Enterprise Edition.
3. Configure the server name and account permissions, and select a pricing plan.
4. On Step 4 in the VM setup wizard, in **SQL Server Settings**, locate **R Services (Advanced Analytics)** 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. R Services 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.

   ```
   execute 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 will be connecting to the instance from a remote data science client, complete additional steps as necessary.

Additional Steps

Some additional steps might be needed to use R Services if you expect remote clients to access the server as a remote SQL Server compute context.

Unblock the firewall

You must change a firewall rule on the virtual machine to ensure that you can access the SQL Server instance from a remote data science client. Otherwise, you might not be able to use compute contexts that require use of the virtual machine's workspace.

By default, the firewall on the Azure virtual machine includes a rule that blocks network access for local R user accounts.

To enable access to R Services 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 R clients calling the server will need to issue ODBC queries as part of their R 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 Set Up SQL Server 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 to SQL Server R Services. 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.

How to disable R Services on an instance

You can also enable or disable the feature on an existing virtual machine at any time.

1. Open the virtual machine blade
2. Click **Settings**, and select **SQL Server configuration**.

Add SQL Server R Services to an existing SQL Server 2016 Enterprise virtual machine

If you created an Azure VM with SQL Server 2016 that did not include R Services, 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 see Set Up SQL Server R Services.
3. (Optional) Configure database access for R worker accounts, if needed for remote script execution. For more information, see Set Up SQL Server R Services.
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.

See Also

Set Up Sql Server R Services

Installing Machine Learning Components without Internet Access

4/29/2017 • 6 min to read • Edit Online

Because the R and Python components provided with SQL Server 2016 or SQL Server 2017 are open source, Microsoft does not install R or Python components by default.

Instead, we provide the related installers and bundled packages as a convenience on the Microsoft Download Center and other trusted sites. You must consent to the appropriate license, and then SQL Server setup will install R or Python components for you.

This topic provides the download locations for the installers and an overview of the offline setup process.

Installation Process

Typically, setup of the machine components used in SQL Server 2016 and SQL Server 2017 requires an Internet connection. When SQL Server setup runs, if you have selected any of the machine learnig options, setup will check for the Python or R installers, as well as any other required components. If there is an Internet connection, SQL Server will install them for you.

> **IMPORTANT**
>
> On a server without Internet access, you must download additional installers before continuing with setup.

At minimum, you will need to download the R or Python installers that are supported for the version or build number of SQL Server that you are installing.

Depending on your server's configuration, you might need additional components, such as .NET Core. See Additional Components for details.

After you have downloaded the installers, you use them when installing the feature as part of SQL Server setup.

Step 1. Obtain additional installers

For **R** in SQL Server 2016 and SQL Server 2017, you'll need to get two different installers. The SQL Server setup wizard will ensure that they are installed in the correct order.

- Installers with **SRO** in the name provide the open source components.
- Insallers with **SRS** in the name contain components provided by Microsoft, including those for database integration.

For **Python** in SQL Server 2017, download the single CAB file, and any prerequisites.

1. Download the installers from the Microsoft Download Center sites onto a computer with Internet access, and save the installer rather than running it.
2. Copy the installer (CAB) files to the computer where you will install machine learning components.
3. Currently, the setup wizard installs English by default. To install using a different language, modify the installer file names as described here: Modifications Required for Different Language Locales.
4. Download any additional components that are required, such as MPI or .NET Core.
5. Optionally, you can download the archived source code for the open source components, but this is not required for SQL Server setup, and can be completed at any time. For more information, see R Server for

Windows.

NOTE

Be sure to get the files that match the version of SQL Server you will be installing.

Support for Python is provided in SQL Server 2017 CTP 2.0. Earlier versions, including SQL Server 2016, do not support Python.

For a step-by-step walkthrough of the offline installation process for R Services in SQL Server 2016, we recommend article by the SQL Server Customer Advisory Team. It also covers patching and slipstream setup scenarios.

Step 2. Run offline setup using the SQL Server setup wizard

1. Run the SQL Server setup wizard.
2. When the setup wizard displays the licensing page, click **Accept**.
3. A dialog box opens that prompts you for the **Install Path** of the required packages.
4. click **Browse** to locate the folder containing the installer files you copied earlier.
5. If the correct files are found, you can click **Next** to indicate that the components are available.
6. Complete the SQL Server setup wizard.
7. Perform the required post-installation steps to make sure the service is enabled.

Downloads

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	
Microsoft R Open	SRO_3.2.2.13000_1033.cab
Microsoft R Server	SRS_8.0.3.13000_1033.cab
SQL Server 2016 SP 1	

RELEASE	DOWNLOAD LINK
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 GDR	
Microsoft R Open	SRO_3.2.2.16000_1033.cab
Microsoft R Server	SRS_8.0.3.16000_1033.cab
SQL Server 2017 CTP 1	
Microsoft R Open	SRO_3.3.0.16000_1033.cab
Microsoft R Server	SRS_9.0.0.16000_1033.cab
SQL Server 2017 CTP 1.1	
Microsoft R Open	SRO_3.3.2.0_1033.cab
Microsoft R Server	SRS_9.0.1.16000_1033.cab
SQL Server 2017 CTP 1.4	
Microsoft R Open	SRO_xxxx_1033.cab
Microsoft R Server	SRS_xxx.xxx_1033.cab
SQL Server 2017 CTP 2.0	
Microsoft R Open	SRO_3.3.3.0_1033.cab
Microsoft R Server	SRS_9.1.0.0_1033.cab
Microsoft Python Open	SPO_9.1.0.0_1033.cab
Microsoft Python Server	SPS_9.1.0.0__1033.cab

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

COMPONENT	VERSION
Microsoft MPI	7.1.12437.25
Microsoft Visual C++ 2013 Redistributable	12.0.30501.0
Microsoft Visual C++ 2015 Redistributable	14.0.23026.0

Installing for Different Language Locales

If you download the .cab files as part of SQL Server setup on a computer with Internet access, the setup wizard detects the local language and automatically changes the language of the installer.

However, if you are installing one of the localized versions of SQL Server to a computer without Internet access and download the R installers to a local share, you must manually edit the name of the downloaded files and insert the correct language identifier for the language you are installing.

For example, if you are installing 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.

Slipstream Upgrades

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 will look for the matching version of the installers for the machine learning components.

Command-line Arguments for Setup

When performing an unattended setup, you will need to provide the following command-line arguments. Note that you do not need to set any additional flags to install additional required components. Prerequisites such as .NET core are installed silently by default.

Location of installers

- `/UPDATESOURCE` to specify the location of the local file containing the SQL Server update installer
- `/MRCACHEDIRECTORY` to specify the folder containing the R component CAB files

R components in SQL Server 2016

- `/ADVANCEDANALYTICS` to get engine support for external scripts
- `/IACCEPTROPENLICENSETERMS="True"` to accept the separate R licensing agreement

R components in SQL Server SQL Server 2017

- `/ADVANCEDANALYTICS` to get engine support for external scripts
- `/SQL_INST_MR` to use R

- `/IACCEPTROPENLICENSETERMS="True"` to accept the separate R licensing agreement

Python components in SQL Server 2017

- `/ADVANCEDANALYTICS` to get engine support for external scripts
- `/SQL_INST_MPY` to use Python
- `/IACCEPTPYTHONLICENSETERMS="True"` to accept the separate R licensing agreement

> **TIP**
>
> 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.

See Also

Install Microsoft R Server

Installing SQL Server R Services on an Azure Virtual Machine

4/19/2017 • 3 min to read • Edit Online

If you deploy an Azure virtual machine that includes SQL Server 2017, you can now select R Services as a feature to be added to the instance when the VM is created.

- Create a new VM with SQL Server 2016 and R Services
- Add R Services to an existing virtual machine with SQL Server 2016

Create a new SQL Server 2016 Enterprise Virtual Machine with R Services Enabled

1. In the Azure Portal, click VIRTUAL MACHINES and then click NEW.
2. Select SQL Server 2016 Enterprise Edition.
3. Configure the server name and account permissions, and select a pricing plan.
4. On Step 4 in the VM setup wizard, in **SQL Server Settings**, locate **R Services (Advanced Analytics)** 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. R Services 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.

   ```
   execute 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 will be connecting to the instance from a remote data science client, complete additional steps as necessary.

Additional Steps

Some additional steps might be needed to use R Services if you expect remote clients to access the server as a remote SQL Server compute context.

Unblock the firewall

You must change a firewall rule on the virtual machine to ensure that you can access the SQL Server instance from a remote data science client. Otherwise, you might not be able to use compute contexts that require use of the virtual machine's workspace.

By default, the firewall on the Azure virtual machine includes a rule that blocks network access for local R user accounts.

To enable access to R Services 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 R clients calling the server will need to issue ODBC queries as part of their R 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 Set Up SQL Server 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 to SQL Server R Services. 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.

How to disable R Services on an instance

You can also enable or disable the feature on an existing virtual machine at any time.

1. Open the virtual machine blade
2. Click **Settings**, and select **SQL Server configuration**.

Add SQL Server R Services to an existing SQL Server 2016 Enterprise virtual machine

If you created an Azure VM with SQL Server 2016 that did not include R Services, 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 see Set Up SQL Server R Services.
3. (Optional) Configure database access for R worker accounts, if needed for remote script execution. For more information, see Set Up SQL Server R Services.
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.

See Also

Set Up Sql Server R Services

Upgrade an Instance of R Services using sqlBindR.exe

4/29/2017 • 7 min to read • Edit Online

When you install the latest version of Microsoft R Server for Windows, it includes a tool that you can use to upgrade the R components associated with an instance of SQL Server 2016 R Services. There are two versions of the tool: a wizard, and a command-line utility

This topic describes how to use these tools to upgrade a compatible instance of SQL Server 2016 with R Services, or to revert an instance that was previously bound.

NOTE

If you are using Machine Learning Services in SQL Server 2017, you do not need to apply this upgrade. Machine learning components are always automatically upgraded to the latest version of R Server at each SQL Server service release.

Upgrade an Instance

The upgrade process is referred to as **binding**, because it changes the support model for SQL Server machine learning components to use the new Modern Lifecycle Policy. However, the upgrade does not change the support model for the SQL Server database.

In general, this licensing system ensures that your data scientists will always be using the latest version of R. For more information about the terms of the Modern Lifecycle Policy, see Support Timeline for Microsoft R Server.

When you bind an instance, several things happen:

- The support policy for R Server and SQL Server R Services is changed from the SQL Server 2016 support policy to the new Modern Lifecycle Policy.
- The R components associated with that instance will be automatically upgraded with each release, in lock-step with the R Server version that is current under the new Modern Lifecycle Policy.
- New packages are added, which are included by default with R Server, such as RODBC, MicrosoftML, olapR, and sqlrutils.
- The instance can no longer be manually updated, except to add new packages.

If you later decide that you want to stop upgrading the instance at each release, you must **unbind** the instance as described in this section, and then uninstall Microsoft R Server components as described in this article: Run Microsoft R Server for Windows. When the process is complete, future R Server upgrades will no longer affect the instance.

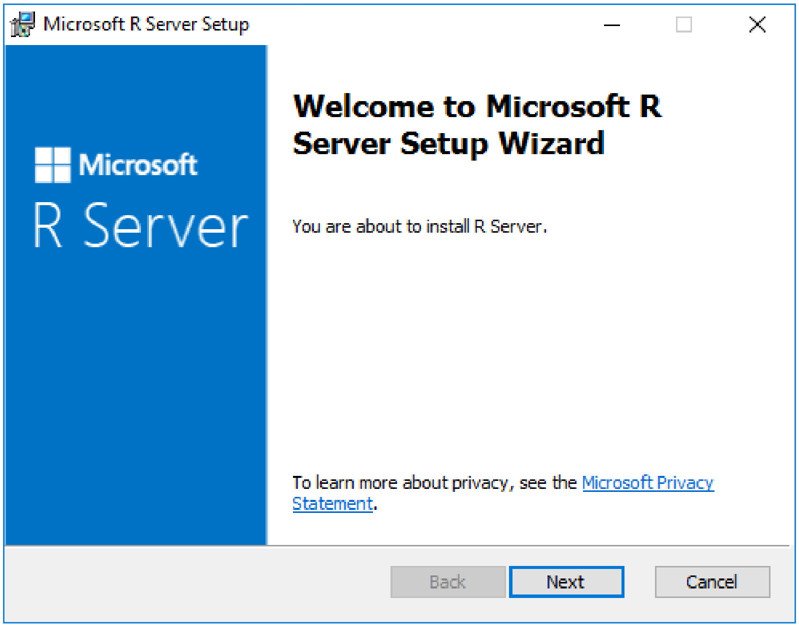
Prerequisites for upgrade

1. Identify instances that are candidates for an upgrade.
   - SQL Server 2016 with R Services installed
   - Service Pack 1 plus CU3
2. Get **R Server**, by downloading the separate Windows installer.

   How to install R Server 9.0.1 on Windows using the standalone Windows installer

Upgrade using the new setup wizard

1. Start the new installer for R Server on the computer that has the instance you want to upgrade.



Microsoft R Server Setup

Welcome to Microsoft R Server Setup Wizard

You are about to install R Server.

To learn more about privacy, see the Microsoft Privacy Statement.

2. Accept the licensing agreement for Microsoft R Server 9.1.0, and click **Next**.
3. Accept the licensing conditions for the R open source components, and click **Next**.
4. On **Select Installation Folder**, accept the defaults, or specify a different location where R libraries will be installed.
5. The installer will identify any local instances that are candidates for binding. If no instances are shown, it means that no valid instances were found. You might need to patch the server, or check whether R Services was installed.
6. Select the check box next to any instance that you want to upgrade, and click **Next**.
7. The process can take a while.

   During the installation, the R libraries used by SQL Server R Services are replaced with the libraries for Microsoft R Server 9.1.0.

   Launchpad is not affected by the process, but the libraries in the R_SERVICES folder will be removed and the properties for the service will be changed, to use the libraries in `C:\Program Files\Microsoft\R Server\R_SERVER`.

Upgrade using the command line

After Microsoft R Server has been installed, you can run just the SqlBindR.exe tool from the command line.

1. Open a command prompt as administrator and navigate to the folder containing sqlbindr.exe. The default location is `C:\Program Files\Microsoft\R Server\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 `MSSQL13.MSSQLSERVER` for the default instance, or something like `SERVERNAME.MYNAMEDINSTANCE`.

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

   For example, to upgrade the default instance, type: `SqlBindR.exe /bind MSSQL13.MSSQLSERVER`

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

Revert or Unbind an Instance

To restore an instance of SQL Server to use the original R libraries installed by SQL Server, you must perform an **unbind** operation. You can do this either by re-running the setup wizard for Microsoft R Server, or by running the

SqlBindR utility from the command line.

When unbinding is complete, the libraries for Microsoft R Server 9.1.0 are removed, and the original R libraries used by SQL Server R Services are restored.

The properties of the SQL Server Launchpad are edited to use the R libraries in the default folder for R_SERVICES, in `C:\Program Files\Microsoft\R Server\R_SERVER`.

Unbind using the wizard

1. Download the new installer for Microsoft R Server 9.1.0.
2. Run the installer on the computer that has the instance you want to unbind.
3. The installer will identify local instances that are candidates for unbinding.
4. Deselect the check box next to the instance that you want to revert to the original SQL Server R Services configuration.
5. Accept the licensing agreement for Microsoft R Server 9.1.0. You must accept the licensing agreement even if you are removing R Server.
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 MSSQL13.MSSQLSERVER
   ```

Known Issues

This section lists known issues specific to use of the SqlBindR.exe utility, or to upgrades using the Microsoft R Server setup utility that affect SQL Server instances.

Restoring packages that were previously installed

In the upgrade utility that was included with Microsoft R Server 9.0.1, the utility did not restore the original packages or R components completely, requiring that the user run repair on the instance, apply all service releases, and then restart the instance.

However, the latest version of the upgrade utility, for Microsoft R Server 9.1.0, will perform the restore of R Services features automatically. Therefore, you do not need to reinstall the R components or re-patch the server. However, you will still need to install any R packages that might have been added after the initial install of SQL Server R Services.

If you have used the package management roles to install and share package, this tasks 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 Installing and Managing R Packages

Cannot perform upgrade from 9.0.1

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 will display a list of all valid instances, and then by default select 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.

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_<guid>`

> **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 will be removed.

sqlbindr.exe command syntax

Usage

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

Parameters

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

Errors

The tool returns the following error messages:

ERROR	RESOLUTION
An error occurred while binding the instance	The instance could not be bound. Contact support for assistance.
The instance is already bound	You ran the *bind* command, but the specified instance is already bound. Choose a different instance.
The instance is not bound	You ran the *unbind* command, but the instance you specified is not bound. Choose a different instance that is compatible.
Not a valid SQL instance ID	You might have typed the instance name incorrectly. Run the command again with the *list* argument to see the available instance IDs.
No instances found	This computer does not have an instance of SQL Server R Services.

ERROR	RESOLUTION
The instance must have a compatible version of SQL R Services (In-Database) installed.	See the compatibility requirements in this topic for details.
An error occurred while unbinding the instance	The instance could not be unbound. Contact support for assistance.
An unexpected error has occurred	Other errors. Contact support for assistance.
No SQL instances found	This computer does not have an instance of SQL Server.

See Also

R Server Release Notes

Install Microsoft R Server from the Command Line

4/29/2017 • 3 min to read • Edit Online

This topic describes how to use SQL Server command-line arguments to install Microsoft R Server in SQL Server 2016, or install Machine Learning Server (Standalone) in SQL Server 2017.

> **NOTE**
>
> You can also install Microsoft R Server by using a separate Windows installer. For more information, see Install R Server 9.0.1 for Windows.

Prerequisites

This method of installation requires that you know how to perform a command-line installation of SQL Server and are familiar with its scripting arguments.

- **Unattended** installation requires that you specify the location of the setup utility, and use arguments to indicate which features to install.
- For a **quiet** installation, provide the same arguments and add the **/q** switch. No prompts will be provided and no interaction is required. However, setup will fail if any required arguments are omitted.

For more information, see Install SQL Server 2016 from the Command Prompt.

SQL Server 2017: Microsoft Machine Learning Server (Standalone)

Run the following command from an elevated command prompt to install only Microsoft Machine Learning Server (Standalone) and its prerequisites. The example shows the arguments used to install R.

```
Setup.exe /q /ACTION=Install /FEATURES=SQL_SHARED_MR, SQL_INST_MR  /IACCEPTROPENLICENSETERMS
/IACCEPTSQLSERVERLICENSETERMS
```

To view progress and prompts, remove the */q* argument.

- **FEATURES = SQL_SHARED_MR** gets only the Machine Learning Server components. This includes any prerequisites, which are installed silently by default.
- **SQL_INST_MR** is required to installl support for the R language.
- **SQL_INST_MPY** is required to install support for Python.
- **IACCEPTROPENLICENSETERMS** indicates you have accepted the license terms for using the open source R components.
- **IACCEPTPYTHONLICENSETERMS** indicates you have accepted the license terms for using the Python components.
- **IACCEPTSQLSERVERLICENSETERMS** is required to run the setup wizard.

Notes

1. The FEATURES argument is required, as is the SQL Server licensing terms.
2. Specify at least one language, together with the licensing agreement flag.
3. You can install one language, or both R and Python, but a separate license is required for each.

SQL Server 2016: Microsoft R Server (Standalone)

Run the following command from an elevated command prompt to install only Microsoft R Server (Standalone) and its prerequisites. The example shows the arguments used with SQL Server 2016 setup.

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

Offline Installation

If you install Machine Learning Server or Microsoft R Server (Standalone) on a computer that has no Internet access, you must download the required R components in advance, and copy them to a local folder. For download locations, see Installing R Components without Internet Access.

What Is Installed

After setup is complete, you can review the configuration file created by SQL Server setup, along with a summary of setup actions.

By default, all setup logs and summaries for SQL Server and related features are created in the following folders:

- SQL Server 2016: `C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log`
- SQL Server 2017: `C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log`

A separate subfolder is created for each feature installed.

To set up another instance of Microsoft R Server with the same parameters, you can re-use the configuration file that is created during installation. For more information, see Install SQL Server Using a Configuration File

Customize Your R Environment

After installation, you can install additional R packages. For more information, see Installing and Managing R Packages.

> **IMPORTANT**
>
> If you intend to run your R code on SQL Server, make sure that you install the same packages on the computer you'll use for developing the solution, and the SQL Server instance where you'll execute or deploy the solution.

After you have installed Machine Learning Services for R (In-Database), you can use the separate Windows installer to upgrade the version of R that is associated with a specified SQL Server instance. For more information, see Use SqlBindR to Upgrade R.

Advanced Analytics Virtual Machines on Azure

6/5/2017 • 5 min to read • Edit Online

Virtual machines on Azure are a convenient option for quickly configuring a complete server environment for machine learning solutions. This topic lists some virtual machine images that contain R Server, SQL Server with machine learning, or other data science tools from Microsoft .

> **TIP**
>
> We recommend that you use the new version of the Azure portal, and the Azure Marketplace. Some images are not available when browsing the Azure Gallery on the classic portal.

The Azure Marketplace contains multiple virtual machines that support data science. This list is not intended to be comprehensive, but only provide the names of images that include either Microsoft R Server or SQL Server machine learning services, to facilitate discovery.

How to Provision a VM

If you are new to using Azure VMs, we recommend that you see these articles for more information about using the portal and configuring a virtual machine.

- Virtual Machines - Getting Started
- Getting Started with Windows Virtual Machines

Find an image

1. From the Azure dashboard, click **Marketplace**.
   - Click **Intelligence and analytics** or **Databases**, and then type "R" in the **Filter** control to see a list of R Server virtual machines.
   - Other possible strings for the **Filter** control are *data science* and *machine learning*
   - Use the % wildcard in search to find VM names that contain a target string, such as *R* or *Julia*.
2. To get R Server for Windows, select **R Server Only SQL Server 2016 Enterprise**.

   R Server is licensed as a SQL Server Enterprise Edition feature, but version 9.1. is installed as a standalone server and serviced under the Modern Lifecycle support policy.

3. After the VM has been created and is running, click the **Connect** button to open a connection and log into the new machine.

4. After you connect, you might need to install additional R tools or development tools.

Install additional R tools

By default, Microsoft R Server includes all the R tools installed with a base installation of R, including RTerm and RGui. A shortcut to RGui has been added to the desktop, if you want to get started using R right away.

However, you might wish to install additional R tools, such as RStudio, the R Tools for Visual Studio (RTVS), or Microsoft R Client. See the following links for download locations and instructions:

- R Tools for Visual Studio
- Microsoft R Client
- RStudio for Windows

After installation is complete, be sure to change the default R runtime location so that all R development tools use the Microsoft R Server libraries.

Configure server for web services

If the virtual machine includes R Server, additional configuration is required to use web service deployment, remote execution, or leverage R Server as a deployment server in your organization. For instructions, see Configure R Server for Operationalization.

> **NOTE**
>
> Additional configuration is not needed if you just want to use packages such as RevoScaleR or MicrosoftML.

R Server Images

Microsoft Data Science Virtual Machine

Comes configured with Microsoft R Server, as well as Python (Anaconda distribution), a Jupyter notebook server, Visual Studio Community Edition, Power BI Desktop, the Azure SDK, and SQL Server Express edition.

The Windows version runs on Windows Server 2012, and contains many special tools for modeling and analytics, including CNTK and mxnet, popular R packages such as xgboost, and Vowpal Wabbit.

Linux Data Science Virtual Machine

Also contains popular tools for data science and development activities, including Microsoft R Open, Microsoft R Server Developer Edition, Anaconda Python, and Jupyter notebooks for Python, R and Julia.

This VM image was recently updated to include JupyterHub, open source software that enables use by multiple users, through different authentication methods, including local OS account authentication, and Github account authentication. JupyterHub is a particularly useful option if you want to run a training class and want all students to share the same server, but use separate notebooks and directories.

Images are provided for Ubuntu, Centos, and Centos CSP.

R Server Only SQL Server 2016 Enterprise

This virtual machine includes a standalone installer for R Server 9.1. that supports the new Modern Software Lifecycle licensing model.

R Server is also available in images for Linux CentOS version 7.2, Linux RedHat version 7.2, and Ubuntu version 16.04.

> **NOTE**
>
> These virtual machines replace the **RRE for Windows Virtual Machine** that was previously available in the Azure Marketplace.

SQL Server Images

To use R Services (In-Database) or Machine Learning Services, you must install one of the SQL Server Enterprise or Developer edition virtual machines, and add the machine learning service, as described here: Installing SQL Server R Services on an Azure Virtual Machine.

> **NOTE**
>
> Currently, machine learning services are not supported on the Linux virtual machines for SQL Server 2017, or in Azure SQL Database. You must use either SQL Server 2016 SP1 or SQL Server 2017 for Windows.

The Data Science Virtual Machine also includes SQL Server 2016 with the R services feature already enabled.

Other VMs

Deep Learning Toolkit for the Data Science Virtual Machine

This virtual machine contains the same machine learning tools available on the Data Science Virtual machine, but with GPU versions of mxnet, CNTK, TensorFlow, and Keras. This image can be used only on Azure GPU N-series instances.

The deep learning toolkit also provides a set of sample deep learning solutions that use the GPU, including image recognition on the CIFAR-10 database and a character recognition sample on the MNIST database. GPU instances are currently available in South Central US, East US, West Europe, and Southeast Asia.

> **IMPORTANT**
>
> GPU instances are currently only available in the South Central US.

Frequently Asked Questions

Can I install a virtual machine with SQL Server 2017?

Images are available that contain SQL Server 2017 CTP 2.0 for Linux environments, but these environments currently do not support Machine Learning Services.

A Windows-based virtual machine for SQL Server 2017 Enterprise Edition that includes Machine Learning Services will be available after public release.

As an alternative, you can use the SQL Server 2016 image, and upgrade your instance of R as described here: Upgrade an Instance using SqlBindR.

Or, create a virtual machine and download the CTP 2.0 preview of SQL Server 2017.

How do I access data in an Azure storage account?

When you need to use data from your Azure storage account, there are several options for accessing or moving the data:

- Copy the data from your storage account to the local file system using a utility, such as AzCopy.
- Add the files to a file share on your storage account and then mount the file share as a network drive on your VM. For more information, see Mounting Azure files.

How do I use data from Azure Data Lake Storage (ADLS)?

You can read data from ADLS storage using RevoScaleR, if you reference the storage account the same way that you would an HDFS file system, by using webHDFS. For more information, see this article: Using R to perform FileSystem Operations on Azure Data Lake Store.

See Also

SQL Server R Services

Setup or Configure R Tools

5/15/2017 • 2 min to read • Edit Online

Microsoft R Server provides all the base R libraries, the a set of ScaleR packages, and standard R tools that you need to develop and test R code. However, if you want to use a dedicated R development environment, there are several available, including many free tools.

Basic R Tools

Additional tools are not required in an installation of Microsoft R Server, because all the standard R tools that are included in a *base installation* of R are installed by default.

- **RTerm**: A command-line tool 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, find the R library location. This varies depending on whether you installed only SQL Server R Services, or if you also installed R Server (Standalone). For more information, see What is Installed and Where to Find R Packages

Then, look in the folder `..\R_SERVER\bin\x64`.

> **TIP**
> Need help with the R tools? Documentation is included, in the setup folder: `C:\Program Files\Microsoft SQL Server\R_SERVER\doc` and in `C:\Program Files\Microsoft SQL Server\R_SERVER\doc\manual`.
>
> Or, just open **RGui**, click **Help**, and select one of the options.

Microsoft R Client

Microsoft R Client is a free download that lets you develop R solutions that can be easily run in Microsoft R Server or in SQL Server R Services. This option is provided to help data scientists who might not have access to R Server (available in Enterprise Edition) develop solutions that use ScaleR.

If you use a different R development environment, such as R Tools for Visual Studio or RStudio, to use ScaleR you must specify that the Microsoft R Client be used as the R executable. This gives you full access to the RevoScaleR package and other features of Microsoft R Server, although performance will be limited.

You can also use the tools provided in R Client, such as RGui and RTerm, to run scripts or write and execute ad hoc R code.

Install Microsoft R Client

R Tools for Visual Studio

For convenience in working with SQL Server databases, consider using R Tools for Visual Studio as your development environment. R Tools for Visual Studio is a free add-in for Visual Studio that works in all editions of Visual Studio. Visual Studio also provides support for Python and F# integration.

For installation instructions, see How to install R Tools for Visual Studio.

> **TIP**
>
> Before you install any new packages, check which R runtime is being used by default. Otherwise it can be very easy to install new R packages to a default library location and then not be able to find them from R Server!

RStudio

If you prefer to use RStudio, some additional steps are required to use the RevoScaleR libraries:

- Install either Microsoft R Server or Microsoft R Client to get the required packages and libraries.
- Update your R path to use the R Server runtime.

For more information, see Configure Your IDE.

See Also

Create a Standalone R Server
Getting Started with Microsoft R Server (Standalone)

Set Up a Data Science Client

4/19/2017 • 1 min to read • Edit Online

After you have configured an instance of SQL Server 2017 by installing **R Services (In-Database)**, you'll want to set up an R development environment that is capable of connecting to the server for remote execution and deployment.

This environment must include the ScaleR packages, and optionally con include a client development environment.

Where to Get ScaleR

Your client environment must include Microsoft R Open, as well as the additional RevoScaleR packages that support distributed execution of R on SQL Server. There are several ways you can install these packages:

- Install Microsoft R Client. Additional setup instructions are provided here: Get Started with Microsoft R Client
- Install Microsoft R Server. You can get Microsoft R Server from SQL Server setup, or by using the new standalone Windows installer. For more information, see Create a Standalone R Server and Introduction to R Server.

If you have an R Server licensing agreement, we recommend the use of Microsoft R Server (Standalone), to avoid limitations on R processing threads and in-memory data.

How to Set Up the R Development Environment

You can use any R development environment of your choice that is compatible with Windows.

- R Tools for Visual Studio supports integration with Microsoft R Open
- RStudio is a popular free environment

After installation, you will need to reconfigure your environment to use the Microsoft R Open libraries by default, or you will not have access to the ScaleR libraries. For more information, see Getting Started with Microsoft R Client.

More Resources

For a detailed walkthrough of how to connect to a SQL Server instance for remote execution of R code, see this tutorial: Data Science Deep Dive: Using the RevoScaleR Packages.

To start using Microsoft R Client and the ScaleR packages with SQL Server, see ScaleR Getting Started.

See Also

Set up SQL Server R Services (In-Database)

Reference for SQL Server Machine Learning APIs

4/19/2017 • 1 min to read • Edit Online

This article provides links to the reference documentation for APIs used by SQL Server R Services and SQL Server 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. To ensure that a single-source is used for documentation, all APIs are documented based on source code and published to the MSDN library.

R

- RevoScaleR functions
- MicrosoftML
- olapR
- sqlrutils
- mrsdeploy functions

"How-to" topics and tutorials 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

Python

revoscalepy

> **NOTE**
>
> This feature is new and still under development. Documentation coming soon.

See Also

RevoPEMAR function reference RevoUtils

Using the MicrosoftML Package with SQL Server R Services

4/29/2017 • 1 min to read • Edit Online

The **MicrosoftML** package that is provided with Microsoft R Server and SQL Server 2017 CTP 1.0 includes multiple machine learning algorithms developed by Microsoft that support multicore processing and fast data streaming. The package also includes transformations for text processing and featurization.

New machine learning algorithms

- **Fast Linear.** 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.
- **Fast Tree.** 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.
- **Fast Forest.** 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.** 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 Net.** A neural network model for binary classification, multiclass classification, and regression. Supports customizable convoluted networks and GPU acceleration using a single GPU.
- **One-Class SVM.** An anomaly detection model based on the SVM method that can be used for binary classification in unbalanced data sets.

Transformation functions

The **MicrosoftML** package also includes the following functions that can be used for transforming data and extracting features.

- `featurizeText()`

  Generates counts of ngrams in a given text string.

  The function includes the options to detect the language used, perform text tokenization and normalization, remove stopwords, and generate features from text.

- `categorical()`

  Builds a dictionary of categories and transforms each category into an indicator vector.

- `categoricalHash()`

  Converts categorical values into an indicator array by hashing the value and using the hash as an index in the bag.

- `selectFeatures()`

  Selects a subset of features from the given variable, either by counting non-default values, or by computing a mutual-information score with respect to the label.

For detailed information about these new features, see MicrosoftML functions.

Support for MicrosoftML in R Services

The **MicrosoftML** package is fully integrated with the data processing pipeline used by the **RevoScaleR** package. Currently, you can use the **MicrosoftML** package in any Windows-based compute context, including SQL Server R Services.

See Also

Introduction to MicrosoftML

RevoScaleR Function Reference

RevoScaleR

4/19/2017 • 1 min to read • Edit Online

ScaleR Functions for Working with SQL Server Data

5/22/2017 • 3 min to read • Edit Online

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 in the MSDN library.

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.

In addition to defining a data source, you can execute DDL statements from R, if you have the necessary permissions on the instance and database.

- RxSqlServerData - Define a SQL Server data source object.

Perform DDL Statements

In addition to defining a data source, you can execute DDL statements from R, if you have the necessary permissions on the instance and database. These functions execute an ODBC call against the database schema.

- rxSqlServerDropTable - Drop a SQL Server table
- rxSqlServerTableExists - 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 - Get the current compute context.
- rxSetComputeContext - Specify which compute context to use.

Use a Data Source

After you have created a data source object, you can open it to get data, or write new data to it. 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.

- rxIsOpen - Check whether a data source is available
- rxOpen - Open a data source for reading
- rxReadNext - Read data from a source
- rxWriteNext - Write data to the target
- rxClose - Close a data source

For more information about working with these functions, including using data sources other than SQL Server, see Getting Started with Microsoft R.

Work with XDF Files

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.

- rxImport - Move data from an ODBC source to the XDF file
- RxXdfData - Create an XDF data object
- RxDataStep - Moves data from an input data source to an output data source, and optionally transforms the data that is written to the output. For more information about the types of transformation that can be applied, see Transforming and Subsetting Data
- rxReadXdf - Reads data from an XDF file into a data frame

For an example of how XDF files are used, see this tutorial: Data Science Deep Dive - Using the ScaleR Functions

See Also

Comparison of Base R and RevoScaleR Functions

Generating an R Stored Procedure for R Code using the sqlrutils Package

4/19/2017 • 4 min to read • Edit Online

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 `registerStoredProcedure`.

    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 `executeStoredProcedure` 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 *executeStoredProcedure* 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 using olapR

4/19/2017 • 4 min to read • Edit Online

How to 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.
   - `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 completed 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

How to run an existing 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

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 *roll up* summary of Internet sales from all countries.
- The WHERE clause is the *slicer axis*. 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)
```

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)
```

Note that 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, be sure to number 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`. Note that 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. Note that items in the return results 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

RESULTS
-> Assembly Components
-> Assembly Components

See Also

Using Data from OLAP Cubes in R

Introducing revoscalepy

5/15/2017 • 3 min to read • Edit Online

revoscalepy is a new library provided by Microsoft to support distributed computing, remote compute contexts, and high-performance algorithms for Python.

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:

- Support multiple compute contexts, remote or local
- Provide functions equivalent to those in RevoScaleR for data transformation and visualization
- Provide Python versions of RevoScaleR machine learning algorithms for distributed or parallel processing
- Improved performance and use of the Intel math libraries

WARNING

Python support is a new feature in SQL Server 2017 and is supported for preview only.

The **revoscalepy** module contains only a subset of the functionality provided in the corresponding **RevoScaleR** package for R.

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 CTP 2.0
- Microsoft Machine Learning Server 9.1.0, using SQL Server 2017 CTP 2.0 setup

Supported Functions and Data Types

This sections lists the Python data types and new Python functions supported in the **revoscalepy** module for the SQL Server 2017 CTP 2.0 release.

Data types

For a list of mappings between SQL and Python data types, see Python Libraries and Data Types.

Data sources and compute contexts

You can get data from any ODBC database, SQL Server, or XDF file, using the data soure functions listed in the following table.

Remote compute contexts supported for this release are local, or in SQL Server 2017.

Functions

The following functions are included in SQL Server 2017 CTP 2.0.

FUNCTION NAME	CATEGORY
`rx_btrees_ex`	analytic
`rx_dforest_ex`	analytic

FUNCTION NAME	CATEGORY
`rx_dtree_ex`	analytic
`rx_lin_mod_ex`	analytic
`rx_logit_ex`	analytic
`rx_predict_ex`	analytic
`rx_summary`	analytic
`RxInSqlServer`	compute context
`RxLocalSeq`	compute context
`RxFileData`	data source
`RxOdbcData`	data source
`RxSqlServerData`	data source
`RxXdfData`	data source
`rx_data_step_ex`	ETL
`rx_import_datasource`	ETL

Need more function help?

If you are new to the idea of remote compute contexts, we recommend that you start by reading about RevoScaleR and how distributed computing works for machine learning.

- View RevoScaleR help

  Locate the corresponding function in R help or in the MSDN library for RevoScaleR.

  > **NOTE**
  >
  > All functions are pre-release versions. Some functions have not been fully tested, and some functions do not have the same level of functionality as the corresponding function in RevoScaleR.

- Use Python help features

  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 included functions with their signatures.

- IntelliSense in Visual Studio

  If you use Python Tools for Visual Studio, you can use Intellisense to get syntax and argument help.

  For more information, see Installing 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.

Examples

You can run code that includes **revoscalepy** functions in two scenarios:

- Execute a Python script from the command line, or from a Python development environment, and call a remote compute context.
- Embed Python code in the stored procedure, [sp_execute_external_script]. (https://docs.microsoft.com/sql/relational-databases/system-stored-procedures/sp-execute-external-script-transact-sql), and call the stored procedure from any client that supports T-SQL.

> **NOTE**
>
> 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. Other versions of SQL Server do not support Python integration.
>
> Open source distributions of Python do not support SQL Server compute contexts. However, you can install Microsoft Machine Learning Server to publish and consume Python applications from Windows without installing SQL Server. For more information, see Create a Standalone R Server

Using remote compute contexts

This example demonstrates how to run Python using an instance of SQL Server as the compute context.

Create a Model using revoscalepy

Using T-SQL

This example demonstrates how to run Python using Python script embedded in a stored procedure.

Run Python using T-SQL

SQL Server Machine Learning Tutorials

5/17/2017 • 7 min to read • Edit Online

Use these tutorials to learn about machine learning in SQL Server 2016 and SQL Server 2017. Topics covered include:

- How to run R or Python from T-SQL
- What are remote and local compute contexts? How to execute R or Python in a SQL Server compute context
- Developing models in R and saving trained models in a SQL Server table
- How to wrap R or code in a stored procedure
- Optimizing R and Python code for a SQL production environment
- Real-world scenarios for embedding machine learning in applications

Prerequisites

To use these tutorials, you must have installed 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.

For more information about setup, see Prerequisites.

Python Tutorials

> **NOTE**
>
> Support for Python is a new feature in SQL Server 2017 (CTP 2.0). Although the feature is in pre-release and not supported for production environments, we invite you to try it out and send feedback.

Learn how to call Python in T-SQL, using the extensibility mechanism pioneered in SQL Server 2016. Now supports Python!

- Running Python in T-SQL

Create a model using rxLinMod, and run it in SQL Server, using the new **revoscalepy** lilbrary from a remote Python terminal.

- Create a Machine Learning Model in Python using revoscalepy

Create a machine learning model to predict demand for a ski rental business, and operationalize that model for day-to-day demand prediction using stored procedures. All code and data provided!

- Build a predictive model with Python

NEW! Build a complete Python solution using T-SQL stored procedures; all Python code supplied included.

- In-Database Python Analytics for SQL Developers

Learn how to deploy a Python model using the latest version of Microsoft Machine Learning Server.

- Deploy and Consume a Python Model

R Tutorials with SQL Server

This section lists tutorials that were developed for SQL Server 2016 R Services. All require Microsoft R, and make extensive use of features in the RevoScaleR package for SQL Server compute contexts.

> **NOTE**
>
> Unless otherwise indicated, these tutorials are expected to work without modification in SQL Server 2017.

Data Science Deep Dive

Learn how to use the functions in the RevoScaleR packages.

- Getting Started with RevoScaleR and SQL Server

Move data between R and SQL, and switch compute contexts to suit a particular task. You will create models and plots and move them between your development environment and SQL 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 Advanced Analytics for the SQL Developer

Build and deploy a complete R solution, using only Transact-SQL tools.

- In-Database Advanced Analytics for SQL Developers (Tutorial)

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 will be supporting 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.

Using R Code in T-SQL

This quick-start covers the basic syntax for using R in Transact-SQL.

- Using R Code in Transact-SQL (SQL Server R Services)

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'll 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.

Developing an End-to-End Advanced Analytics Solution

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.

- Data Science End-to-End Walkthrough

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. You'll need to download the New York City taxi dataset using PowerShell. Access to a server with SQL Server R Services or Machine Learning Services with R already enabled. For setup help, see Prerequisites.

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.

Predicting customer demand using R in 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.

- Build an intelligent app with SQL Server and R

Customer segmentation using K-Means Clustering

Use unsupervised learning to segment customers based on sales data. (English only)

- Clustering in SQL Server R Services

Learn RevoScaleR

Wondering what RevoScaleR offers that R doesn't? See these tutorials

This R Server quick-start demonstrates how you can write R code once, and deploy anywhere, using RevoScaleR data sources and remote compute contexts.

- Explore R and Scale R in 25 Short Functions

Learn how to use RevoScaleR for advanced modeling and data transformation, and optimize for different compute contexts.

- Diving Deep into Data Analysis

Customizable End-to-End Solutions

The Microsoft Data Science Team has provided a number of solution templates that can be used for copy-paste creation of solutions for common scenarios. All T-SQL and R 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.

Resources and Reading

- Want to know the real story behind R Services? Read this article from the development and PM team: Why did we build it?

- The original SQL Server 2016 Product Samples, available on Github and on the Microsoft Download Center, contains some datasets and code samples for R Services, including a demo of insurance fraud detection based on Benford's law. To get only the samples for R Services (In-Database), select the zip file, and open the folder **Advanced Analytics**. The setup instructions are for earlier releases and should be disregarded.

Prerequisites

SQL Server 2016

To run any of these tutorials, you must download and install **R Services (in-Database)** as described here: Set up SQL Server R Services

SQL Server 2017

In SQL Server 2017 CTP 2.0, R Services has been renamed **Machine Learning Services (in-Database)**. With this latest release, 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:

- Enable the external script execution feature by running `sp_configure 'enable external script', 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: Upgrade and Installation of SQL Server R Services

If you do not already have a preferred R development environment, you can install one of these tools to get started:

- Microsoft R Client
- R Tools for Visual Studio

Note that standard R libraries are insufficient to use these tutorials; both your R development environment and the SQL Server computer running R must have the R packages provided by Microsoft. For more information about what's in Microsoft R, see this article: Microsoft R Products.

See Also

Getting Started with SQL Server R Services

SQL Server Machine Learning Tasks

Run Python Using T-SQL

4/19/2017 • 2 min to read • Edit Online

This example shows how you can run a simple Python script in SQL Server, by using the stored procedure sp_execute_external_script.

Step 1. Create the test data table

Run the following T-SQL statement to create a mapping table for days of the week.

```
CREATE TABLE PythonTest (
    [DayOfWeek] varchar(10) NOT NULL,
    [Amount] float NOT NULL
    )
GO

INSERT INTO PythonTest VALUES
('Sunday', 10.0),
('Monday', 11.1),
('Tuesday', 12.2),
('Wednesday', 13.3),
('Thursday', 14.4),
('Friday', 15.5),
('Saturday', 16.6),
('Friday', 17.7),
('Monday', 18.8),
('Sunday', 19.9)
GO
```

Step 2. Run the Hello World script

The following code loads the Python executable, passes the input data, and for each row of input data, updates the day name in the table with a number representing the day-of-week index.

The section containing the message "Hello World" prints two times because the value of @RowsPerRead is set to 5; therefore, the rows in the table must be processed in two calls to Python.

Note that the Python Data Analysis Library, or **pandas**, is required for passing data to SQL Server, and is included by default with Machine Learning Services.

```
DECLARE @ParamINT INT = 1234567
DECLARE @ParamCharN VARCHAR(6) = 'INPUT '

print '------------------------------------------------------------------------'
print 'Output parameters (before):'
print FORMATMESSAGE('ParamINT=%d', @ParamINT)
print FORMATMESSAGE('ParamCharN=%s', @ParamCharN)

print 'Dataset (before):'
SELECT * FROM PythonTest

print '------------------------------------------------------------------------'
print 'Dataset (after):'
DECLARE @RowsPerRead INT = 5

execute sp_execute_external_script
@language = N'Python',
@script = N'
import sys
import os
print("*******************************")
print(sys.version)
print("Hello World")
print(os.getcwd())
print("*******************************")
if ParamINT == 1234567:
       ParamINT = 1
else:
       ParamINT += 1

ParamCharN="OUTPUT"
OutputDataSet = InputDataSet

global daysMap

daysMap = {
       "Monday" : 1,
       "Tuesday" : 2,
       "Wednesday" : 3,
       "Thursday" : 4,
       "Friday" : 5,
       "Saturday" : 6,
       "Sunday" : 7
       }

OutputDataSet["DayOfWeek"] = pandas.Series([daysMap[i] for i in OutputDataSet["DayOfWeek"]], index =
OutputDataSet.index, dtype = "int32")
',
@input_data_1 = N'SELECT * FROM PythonTest',
@params = N'@r_rowsPerRead INT, @ParamINT INT OUTPUT, @ParamCharN CHAR(6) OUTPUT',
@r_rowsPerRead = @RowsPerRead,
@paramINT = @ParamINT OUTPUT,
@paramCharN = @ParamCharN OUTPUT
with result sets (("DayOfWeek" int null, "Amount" float null))

print 'Output parameters (after):'
print FORMATMESSAGE('ParamINT=%d', @ParamINT)
print FORMATMESSAGE('ParamCharN=%s', @ParamCharN)
GO
```

Expected Results

The stored procedure returns the original data, applies the script, and then returns the modified data.

DAYOFWEEK (BEFORE)	AMOUNT	DAYOFWEEK (AFTER)
Sunday	10	7
Monday	11.1	1
Tuesday	12.2	2
Wednesday	13.3	3
Thursday	14.4	4
Friday	15.5	5
Saturday	16.6	6
Friday	17.7	5
Monday	18.8	
Sunday	19.9	7

Messages

Status messages or errors returned to the Python console are returned as messages in the Query window.

Output parameters (before): ParamINT=1234567 ParamCharN=INPUT Dataset (before):

(10 row(s) affected)

Dataset (after): STDOUT message(s) from external script: C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-packages\revoscalepy

3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul 5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] Hello World C:\PROGRA~1\MICROS~2\MSSQL1~1.MSS\MSSQL\EXTENS~1\MSSQLSERVER01\7A70B3FB-FBA2-4C52-96D6-8634DB843229

3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul 5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] Hello World C:\PROGRA~1\MICROS~2\MSSQL1~1.MSS\MSSQL\EXTENS~1\MSSQLSERVER01\7A70B3FB-FBA2-4C52-96D6-8634DB843229

(10 row(s) affected) Output parameters (after): ParamINT=2 ParamCharN=OUTPUT

Use Python with revoscalepy to Create a Model

4/29/2017 • 2 min to read • Edit Online

This code sample demonstrates how you can create a logistic regression model using one of the algorithms in the **revoscalepy** package for Python, using Microsoft Machine Learning Services.

The **revoscalepy** package for Python contains objects, transformation, and algorithms similar to those provided for the R language's **RevoScaleR** package. With this library, you can create a compute context, 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?

Prerequisites

> **IMPORTANT**
>
> To run Python code in SQL Server, you must have installed SQL Server 2017 CTP 2.0, with the feature, **Machine Learning Services** with Python. Other versions of SQL Server do not support Python integration.

To run this code, execute the sample as a Python script from the command line, or by using a Python development environment that includes the Python integration components provide in this release.

Sample Code

This sample contains the following steps:

1. Import the libraries and functions you need
2. Create the connection to SQL Server and create data source objects for working with the data
3. In your Python code, modify the data so that it can be used by the rxLinMod algorithm
4. Call rxLinMod and define the formula to train the model
5. Generate a set of predictions based on the original data set
6. Create a summary based on the predicted values

All operations are performed using an instance of SQL Server as the compute context.

In general, the process of calling Python in a remote compute context is very much like that used for using R in a remote compute context.

```
from revoscalepy.computecontext.RxComputeContext import RxComputeContext
from revoscalepy.computecontext.RxInSqlServer import RxInSqlServer
from revoscalepy.computecontext.RxInSqlServer import RxSqlServerData
from revoscalepy.functions.RxLinMod import rx_lin_mod_ex
from revoscalepy.functions.RxPredict import rx_predict_ex
from revoscalepy.functions.RxSummary import rx_summary
from revoscalepy.utils.RxOptions import RxOptions
from revoscalepy.etl.RxImport import rx_import_datasource


import os

def test_linmod_sql():
    sqlServer = os.getenv('RTEST_SQL_SERVER', '.')

    connectionString = 'Driver=SQL Server;Server=' + sqlServer +
';Database=RevoTestDb;Trusted_Connection=True;'
    print("connectionString={0!s}".format(connectionString))

    dataSource = RxSqlServerData(
        sqlQuery = "select top 10 * from airlinedemosmall",
        connectionString = connectionString,
        colInfo = {
            "ArrDelay" : { "type" : "integer" },
            "DayOfWeek" : {
                "type" : "factor",
                "levels" : ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
            }
        })

    computeContext = RxInSqlServer(
        connectionString = connectionString,
        numTasks = 1,
        autoCleanup = False
        )

    #
    # import data source to avoid factor levels
    #
    data = rx_import_datasource(dataSource)
    print(data)

    #
    # run linmod
    #
    linmod = rx_lin_mod_ex("ArrDelay ~ DayOfWeek", data = data, compute_context = computeContext)
    assert (linmod is not None)
    assert (linmod._results is not None)
    print(linmod)

    #
    # predict results
    #
    data = rx_import_datasource(dataSource)
    del data["ArrDelay"]
    predict = rx_predict_ex(linmod, data = data)
    assert (predict is not None)
    print(predict._results)

    #
    # do a summary
    #
    summary = rx_summary("ArrDelay ~ DayOfWeek", data = dataSource, compute_context = computeContext)
    assert (summary is not None)
    print(summary)

test_linmod_sql()
```

See Also

Deploy and Consume Python Models

In-Database Python Analytics for SQL Developers

6/1/2017 • 2 min to read • Edit Online

The goal of this walkthrough is to provide SQL programmers with hands-on experience building a machine learning solution in SQL Server. In this walkthrough, you'll learn how to incorporate Python into an application by adding Python code to stored procedures.

> **NOTE**
>
> Prefer R? See this tutorial, which provides a similar solution but uses R, and can eb run in either SQL Server 2016 or SQL Server 2017.

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, such as these Python tools:

- PyCharm, a popular IDE
- Spyder, which is included with Visual Studio 2017 if you install the Data Science workload
- Python Extensions for Visual Studio.

After you have created and tested the solution in the IDE, you can deploy the Python code to SQL Server using Transact-SQL stored procedures in the familiar environment of Management Studio.

In this walkthrough, we'll assume that you have been given all the Python code needed for the solution, and you'll focus on building and deploying the solution using SQL Server.

- 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

  Perform basic data exploration and visualization, by calling Python from Transact-SQL stored procedures.

- Step 4: Create Data Features using T-SQL

  Create new data features using custom SQL functions.

- Step 5: Train and Save a Model using T-SQL

  Build and save the machine learning model, using Python in stored procedures.

- Step 6: Operationalize the Model

  After the model has been saved to the database, call the model for prediction using Transact-SQL.

NOTE

We recommend that you do not use SQL Server Management Studio to write or test Python 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.

Scenario

This walkthrough uses the well-known NYC Taxi data set. To make this walkthrough quick and easy, the data is sampled. Using this data, 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.

Requirements

This walkthrough is intended for users who are already familiar with fundamental database operations, such as creating databases and tables, importing data into tables, and creating SQL queries.

All Python code is provided. 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.

Before starting the walkthrough, you must complete these preparations:

- Install an instance of SQL Server 2017 with Machine Learning Services and Python enabled (requires CTP 2.0 or later).
- 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.

Next Step

Step 1: Download the Sample Data

See Also

Machine Learning Services with Python

Step 1: Download the Sample Data

5/26/2017 • 1 min to read • Edit Online

In this step, you'll download the sample dataset and the scripts. 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 the Data and Scripts

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 **TempPythonSQL**.

   ```
   $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:\tempPythonSQL'
   ```

   If the folder you specify in *DestDir* does not exist, it will be created by the PowerShell script.

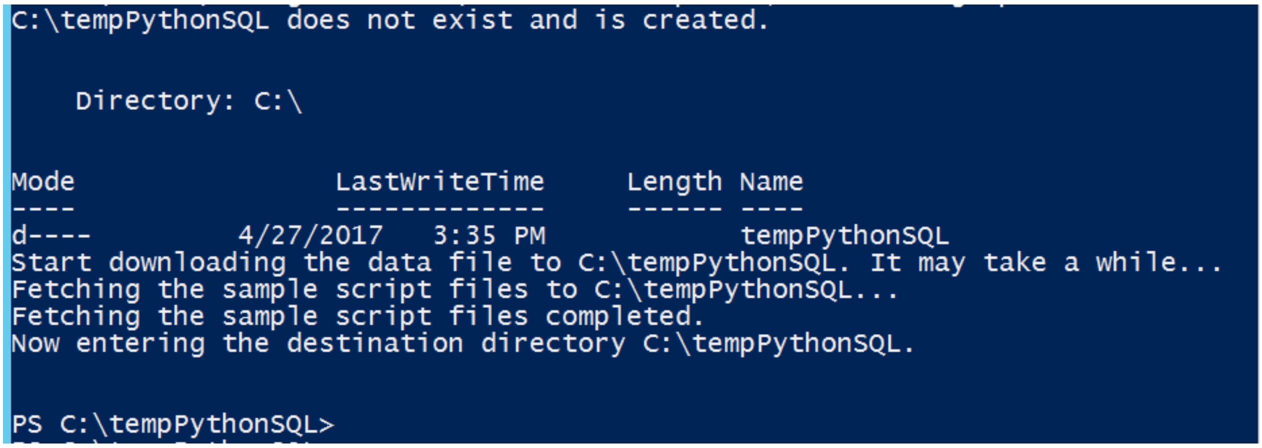
   If you get an error, you can temporarily set the policy for execution of PowerShell scripts to **unrestricted** only for this walkthrough, by using the **Bypass** argument and scoping the changes to the current session. Running this command does not result in a configuration change.

   `Set\-ExecutionPolicy Bypass \-Scope Process`

3. Depending on your Internet connection, the download might take a while. 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.

   ```
   ls
   ```

   Results:



```
C:\tempPythonSQL does not exist and is created.


    Directory: C:\


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----          4/27/2017   3:35 PM            tempPythonSQL
Start downloading the data file to C:\tempPythonSQL. It may take a while...
Fetching the sample script files to C:\tempPythonSQL...
Fetching the sample script files completed.
Now entering the destination directory C:\tempPythonSQL.


PS C:\tempPythonSQL>
```

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

6/6/2017 • 4 min to read • Edit Online

In this step, you'll run one of the downloaded scripts, to create the database objects required for the walkthrough. The script also creates most of the stored procedures you'll use, and uploads the sample data to a table in the database you specified.

Create SQL Objects and Data

Among the downloaded files you should see a PowerShell script. To prepare the environment for the walkthrough, you'll run this script. 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.

Run the script

1. Open a PowerShell command prompt as administrator and run the following command.

    ```
    .\RunSQL_SQL_Walkthrough.ps1
    ```

    You will be prompted to input 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 upload 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:\tempRSQL\nyctaxi1pct.csv`

    > **NOTE**
    >
    > Make sure that your xmlrw.dll is in the same folder as your bcp.exe. If not, please copy it over.

2. As part of this step, all the Transact-SQL scripts are also modified to replace placeholders with the database name and user name that you provide as script inputs.

3. Take a minute to review the stored procedures and functions created by the script.

SQL SCRIPT FILE NAME	FUNCTION

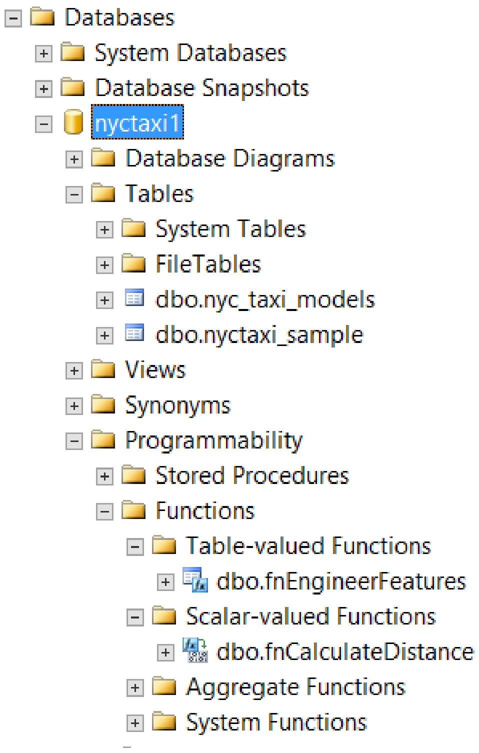
SQL SCRIPT FILE NAME	FUNCTION
create-db-tb-upload-data.sql	Creates a database and two tables: nyctaxi_sample: Contains the main NYC Taxi dataset. A clustered columnstore index is added to the table to improve storage and query performance. The 1% sample of the NYC Taxi dataset will be inserted into this table. nyc_taxi_models: Used to persist the trained advanced analytics model.
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.

4. You'll create some additional stored procedures in the latter part of this walkthrough:

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 (revoscalepy). 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.

5. Log in to the SQL Server instance using SQL Server Management Studio and the login you specified, to verify that you can see the database, tables, functions, and stored procedures that were created.



```
⊟ Databases
    ⊞ System Databases
    ⊞ Database Snapshots
    ⊟ nyctaxi1
        ⊞ Database Diagrams
        ⊟ Tables
            ⊞ System Tables
            ⊞ FileTables
            ⊞ dbo.nyc_taxi_models
            ⊞ dbo.nyctaxi_sample
        ⊞ Views
        ⊞ Synonyms
        ⊟ Programmability
            ⊞ Stored Procedures
            ⊟ Functions
                ⊟ Table-valued Functions
                    ⊞ dbo.fnEngineerFeatures
                ⊟ Scalar-valued Functions
                    ⊞ dbo.fnCalculateDistance
                ⊞ Aggregate Functions
                ⊞ System Functions
```

> **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. Follow instructions here to enable external scrip services.

Next Step

Step 3: Explore and Visualize the Data

Previous Step

Step 1: Download the Sample Data

See Also

Step 3: Explore and Visualize the Data

6/1/2017 • 5 min to read • Edit Online

Developing a data science solution usually includes intensive data exploration and data visualization. In this step, you'll explore the sample data and generate some plots. Later, you'll learn how to serialize graphics objects in Python, and then deserialize those object and make plots.

> **NOTE**
>
> This walkthrough demonstrates only the binary classification task; you are welcome to try building separate models for the other two machine learning tasks, regression and multiclass classification.

Review the Data

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.
- 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 Plots using Python in T-SQL

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 a popular library that includes many functions for creating histograms, scatter plots, box plots, and other data exploration graphs.

In this section, you'll learn how to work with plots using stored procedures. You'll store the `plot` Python object as a **varbinary** data type, and save the plots generated on the server.

Storing plots as varbinary data type

The **revoscalepy** module included with SQL Server 2017 Machine Learning Services contains libraries analogous to the R libraries in the RevoScaleR package. In this example, you'll use the Python equivalent of `rxHistogram` to plot a histogram based on data from a Transact-SQL query. To make it easier, you will wrap it in a stored procedure, *PlotHistogram*.

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.

Create the stored procedure Plots_Python

1. In SQL Server Management Studio, open a new **Query** window.

2. Select the database for the walkthrough, and create the procedure using this statement. Be sure to modify the code to use the correct table name, if needed.

```
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
```

Notes:

- The variable `@query` defines the query text (`'SELECT tipped FROM nyctaxi_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 an Python **pandas** DataFrame for output.

Output varbinary data to viewable graphics file

1. In Management Studio, run the following statement:

```
EXEC [dbo].[SerializePlots]
```

Results

plot 0xFFD8FFE000104A4649... 0xFFD8FFE000104A4649... 0xFFD8FFE000104A4649... 0xFFD8FFE000104A4649...

2. On the client machine, run the following Python code, replacing the server name, database name, and credentials as appropriate.

   For SQL server authentication:

~~~
```python
import pyodbc
import pickle
import os
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER={SERVER_NAME};DATABASE={DB_NAME};UID={USER_NAME};PWD=
{PASSOWRD}')
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())
```
~~~

   For Windows authentication:

~~~
```python
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())
```
~~~

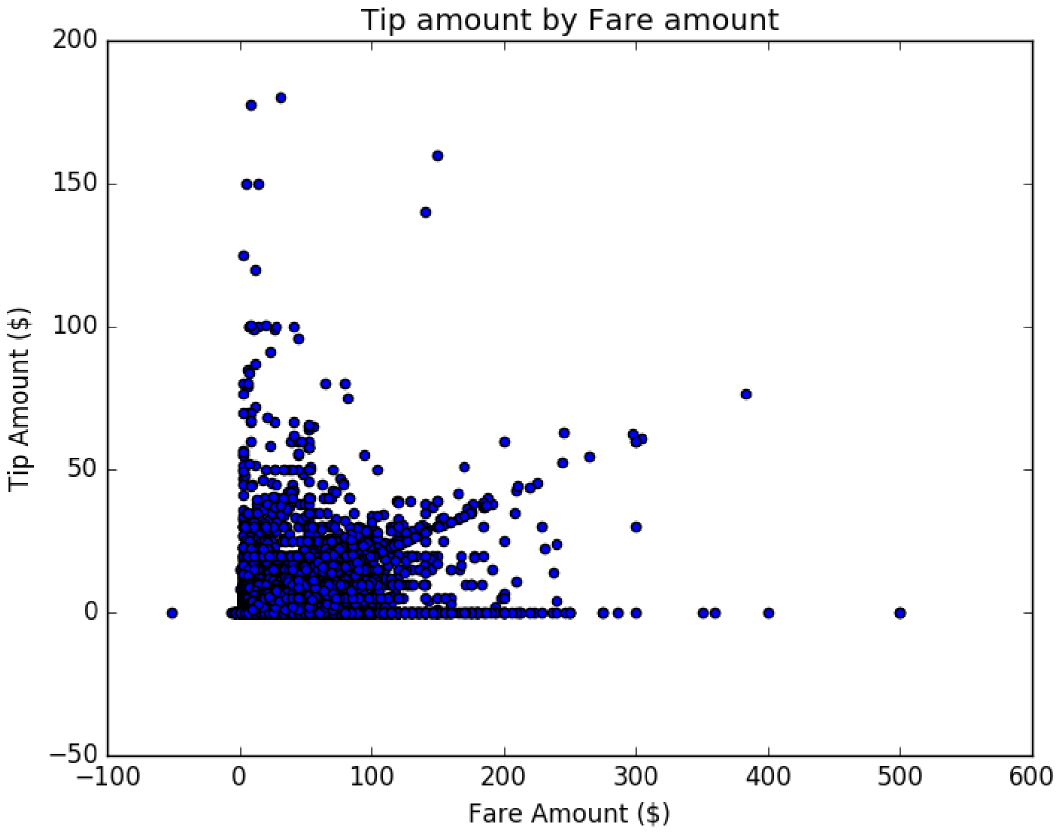
   NOTE

   Make sure the Python version is the same on the client and the server. Also, make sure that the Python libraries you're using on your client (such as matplotlib) are of the same or higher version relative to the libraries installed on the server.

3. If the connection is successful, you will see the results below

   The plots are saved in directory: xxxx

4. The output file will be created in the Python working directory. To view the plot, just open the Python working directory. The following image shows an example 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

See Also

Machine Learning Services with Python

Step 4: Create Data Features using T-SQL

6/1/2017 • 3 min to read • Edit Online

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.

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 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 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 Model using T-SQL

Previous Step

Step 3: Explore and Visualize the Data

See Also

Machine Learning Services with Python

Step 5: Train and Save a Model using T-SQL

6/1/2017 • 5 min to read • Edit Online

In this step, you'll 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, so you can load the modules and call the necessary functions from within a stored procedure. You'll train the model using the data features you just created, and then save the trained model in a SQL Server table.

Split the sample data into training and testing sets

1. Run the following T-SQL commands to create a stored procedure that will divide the data in the nyctaxi_sample table into two parts: nyctaxi_sample_training and nyctaxi_sample_testing.

```
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. 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. Training and testing data are stored in two separate tables.

```
EXEC TrainTestSplit 60
GO
```

Build a logistic regression model using scikit-learn

In this section, you'll create a stored procedure that can be used to train a model using the training data you just prepared. This stored procedure defines the input data and uses a **scikit-learn** function to train a logistic regression model. You call the Python runtime that is installed with SQL Server by using the system stored procedure, sp_execute_external_script.

To make it easier to retrain the model, you can wrap the call to sp_execute_exernal_script in another stored procedure, and pass in the new training data as a parameter. This section will walk you through that process.

1. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure *TrainTipPredictionModelSciKitPy*. Note that 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
  import pandas
  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....

Build a logistic model using the *revoscalepy* package

Now you'll create a different stored procedure that can uses the new **revoscalepy** package to train a logistic regression model. The **revoscalepy** package for Python contains objects, transformation, and algorithms similar to those provided for the R language's **RevoScaleR** package. With this library, you can create a compute context, 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*. This model will be based on the training data you just prepared. 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
import pandas
from revoscalepy.functions.RxLogit import rx_logit_ex

## Create a logistic regression model using rx_logit_ex function from revoscalepy package
logitObj = rx_logit_ex("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 nyctaxi_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:

   - Train a logistic regression model using revoscalepy package on nyctaxi_sample_training data.
   - 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 Python input variable, `InputDataset`.
   - The Python script calls the revoscalepy's LogisticRegression function, which is 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, contained in the Python variable `logitObj`, is serialized and put as an output parameter SQL Server. That output will be inserted into the database table *nyc_taxi_models*, along with its name, as a new row, so that you can retrieve and use it for future predictions.

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'll use the trained models to create predictions.

Next Step

Step 6: Operationalize the Model

Previous Step

Step 4: Create Data Features using T-SQL

See Also

Machine Learning Services with Python

Step 6: Operationalize the Model

6/1/2017 • 7 min to read • Edit Online

In this step, you'll learn to *operationalize* the models that you trained and saved in the previous step. Operationalize in this case means "deploying the model to production for scoring". This is easy to do if your Python code is contained in a stored procedure. You can then call the stored procedure from applications, to make predictions on new observations.

You'll learn two methods for calling a Python model from a stored procedure:

- **Batch scoring mode**: Use a SELECT query to provide multiple rows of data. 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.

Scoring using the scikit-learn model

The stored procedure *PredictTipSciKitPy* uses the scikit-learn model. This stored procedure illustrates the basic syntax for wrapping a Python prediction call in a stored procedure.

- The name of the model to use is provided as input parameter to the stored procedure.
- The stored procedure will then load 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`.
- This data frame is passed to the `predict_proba` function of the logistic regression model, `mod`, which was created by using scikit-learn model.
- 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.
- The stored procedure also calculates an accuracy metric, AUC (area under curve). Accuracy metrics such as AUC can only be generated if you also provide the target label (i.e. 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 simply return the tip probabilities from the features (variable `X` in the stored procedure).

```
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;
        import pandas;
        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
```

Scoring using the revoscalepy model

The stored procedure *PredictTipRxPy* uses a model that was created using the **revoscalepy** library. It works much the same way as the *PredictTipSciKitPy* procedure, but with some changes for the **revoscalepy** functions.

```
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;
      import pandas;
      from sklearn import metrics
      from revoscalepy.functions.RxPredict import rx_predict_ex;

      mod = pickle.loads(lmodel2)
      X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs", "direct_distance"]]
      y = numpy.ravel(InputDataSet[["tipped"]])

      probArray = rx_predict_ex(mod, X)
      probList = []
      for i in range(len(probArray._results["tipped_Pred"])):
        probList.append((probArray._results["tipped_Pred"][i]))

      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
```

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

In this section you'll learn how to pass those arguments to the stored procedure to easily change both the model and the data used for scoring.

1. Define the input data and call the stored procedures for scoring as follows. This example uses the stored procedure PredictTipSciKitPy for scoring, and passes in the model's name and query string

   ```
   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 in the model name and query string.

   ```
   EXEC [dbo].[PredictTipRxPy] 'revoscalepy_model', @query_string;
   ```

Score Individual Rows using scikit-learn model

Sometimes, instead of batch scoring, you might want to pass in a single case, getting 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'll learn how to create single predictions by calling a stored procedure.

1. Take a minute to review the code of the stored procedures PredictTipSingleModeSciKitPy and PredictTipSingleModeRxPy, which are included as part of the download. These stored procedures use the scikit-learn and revoscalepy models, and perform scoring as follows:

   - The model's name and multiple single values are provided as input. These inputs include passenger count, trip distance, and so forth.
   - A table-valued function, `fnEngineerFeatures`, takes input values and converts the latitudes and longitudes to direct distance. Lesson 4 contains a description of this table-valued function.
   - If you call the stored procedure from an external application, ensure that the input data matches the required input features of the Python model. This might include casting or converting the input data to a Python data type, or validating data type and data length.
   - The stored procedure creates a score based on the stored Python model.

PredictTipSingleModeSciKitPy

Here is the definition 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;
      import pandas;

      # 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

Here is the definition of the stored procedure that 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;
      import pandas;
      from revoscalepy.functions.RxPredict import rx_predict_ex;

      # 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_ex(mod, X)

      probList = []
      probList.append(probArray._results["tipped_Pred"])

      # 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
```

1. To try it out, open a new **Query** window, and call the stored procedure, typing parameters for each of the feature columns.

```
-- Call stored procedure PredictTipSingleModeSciKitPy to score using SciKit-Learn model
EXEC [dbo].[PredictTipSingleModeSciKitPy] 'linear_model', 1, 2.5, 631, 40.763958,-73.973373,
40.782139,-73.977303
-- Call stored procedure PredictTipSingleModeRxPy to score using revoscalepy model
EXEC [dbo].[PredictTipSingleModeRxPy] 'revoscalepy_model', 1, 2.5, 631, 40.763958,-73.973373,
40.782139,-73.977303
```

The seven values are for these feature columns, in order:

- *passenger_count*
- *trip_distance*
- *trip_time_in_secs*
- *pickup_latitude*
- *pickup_longitude*
- *dropoff_latitude*
- *dropoff_longitude*

2. The output from both procedures is a probability of a tip being paid for the taxi trip with the above parameters or features.

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 6: Operationalize the Model

See Also

Machine Learning Services with Python

Publish and consume Python web services

4/29/2017 • 18 min to read • Edit Online

Applies to: SQL Server 2017 CTP 2.0 (Public Preview)

You can deploy a working Python solution to a web service by using the operationalization feature in Microsoft Machine Learning Server.

This topic describes the steps to follow, to successfully publish and then run your solution.

The target audience for this article is data scientists who want to learn how to publish Python code or models as web services hosted in Microsoft Machine Learning Server. The article also explains how applications can consume the the code or models. This article assumes that you are proficient in Python.

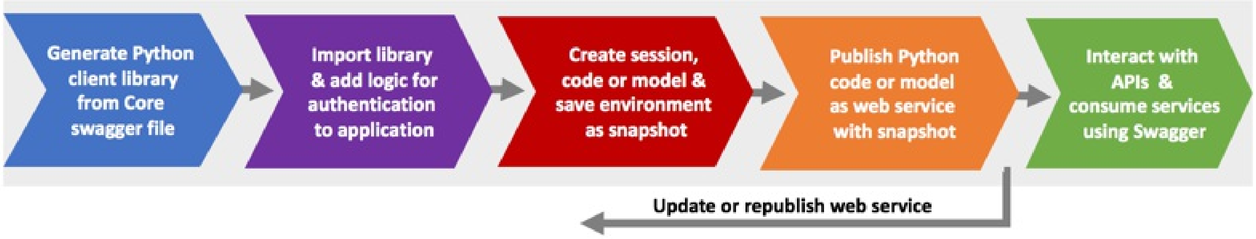
IMPORTANT

Python web services are available only with Python-enabled installs of SQL Server 2017 CTP 2.0 (Public Preview). Availability of this feature will be expanded in future releases.

Overview of workflow

The workflow from publishing to consuming a Python web service can be summarized as follows:

1. Fulfill the prerequisite of generating the Python client library from the core API Swagger document.
2. Add authentication and header logic to your Python script.
3. Create a Python session, prepare the environment, and create a snapshot to preserve the environment.
4. Publish the web service and embed this snapshot.
5. Try out the web service by consuming it in your session.
6. Manage these services.


Generate Python client library from Core swagger file
Import library & add logic for authentication to application
Create session, code or model & save environment as snapshot
Publish Python code or model as web service with snapshot
Interact with APIs & consume services using Swagger
Update or republish web service

This article discusses each step of the workflow, and includes sample Python code using the iris dataset.

Sample Code

This sample code assumes you have satisfied the prerequisites to generate a Python client library from that Swagger file and that you've used Autorest.

After the code block, you'll find a step-by-step walkthrough with more detailed description of each steo in the process.

IMPORTANT

This example uses the local `admin` account for authentication. However, you should substitute the credentials and authentication method configured by your administrator.

```
####################################################
##       IMPORT GENERATED CLIENT LIBRARY        ##
####################################################

# Import the generated client library.
import deployrclient

####################################################
##              AUTHENTICATION                  ##
####################################################

#Using client library generated from Autorest
#Create client instance and point it at an R Server.
#In this case, R Server is local.
client = deployrclient.DeployRClient("http://localhost:12800")

#Define the login request and provide credentials
#Update values with the connection parameters from your admin
login_request = deployrclient.models.LoginRequest("<<your-username>>","<<your-password>>")

#Make a call to the /login API.
#Store the returned an access token in a variable.
token_response = client.login(login_request)

#Add the returned access token to the headers variable.
headers = {"Authorization": "Bearer {0}".format(token_response.access_token)}

#Verify that the server is running.
#Remember to include `headers` in every request!
status_response = client.status(headers)
print(status_response.status_code)


####################################################
##        CREATE SESSION, MODEL, SNAPSHOT       ##
####################################################

#Since already logged in, create a Python session.
#Define session using name (`Session 1`) and `runtime_type="Python"`.
#Remember to specify the Python runtime type.
create_session_request = deployrclient.models.CreateSessionRequest("Session 1", runtime_type="Python")

#Make the call to start the session.
#Remember to include headers in every method call to the server.
#Returns a session ID.
response = client.create_session(create_session_request, headers)

#Store the session ID in a variable called `session_id`
#to be able to identify it later at execution time.
session_id = response.session_id

#Create model - Import SVM and datasets from the SciKit-Learn library
execute_request = deployrclient.models.ExecuteRequest("from sklearn import svm\nfrom sklearn import datasets")
execute_response = client.execute_code(session_id,execute_request, headers)
#Report if it was a success
execute_response.success

#Define the untrained Support Vector Classifier (SVC) object
#and the dataset to be preloaded.
execute_request = deployrclient.models.ExecuteRequest("clf=svm.SVC()\niris=datasets.load_iris()\nnames={0:'I.
setosa',1:'I. versicolor',2:'I. virginica'}")
```

```
setosa',1: 'I. versicolor',2: 'I. virginica'})
#Now, go create the object and preload Iris Dataset in R Server
execute_response = client.execute_code(session_id,execute_request, headers)
#Report if it was a success
execute_response.success

#Define two rows from the Iris Dataset as a sample for scoring
workspace_object = deployrclient.models.WorkspaceObject("species_1",[7,3.2,4.7,1.4])
workspace_object_2 = deployrclient.models.WorkspaceObject("species_2",[3,2.6,3,2.5])

#Define how to train the classifier model; what to predict; what to return
execute_request = deployrclient.models.ExecuteRequest(
    "clf.fit(iris.data, iris.target)\n"+
    "result=clf.predict(species_1)\n"+
    "other_result=clf.predict(species_2)",
    [workspace_object,workspace_object_2], #Input
    ["result", "other_result"]) #Output

#Now, go train that model on the Iris Dataset in R Server
execute_response = client.execute_code(session_id,execute_request, headers)

#If successful, print name and result of each output parameter. Else, print error.
if(execute_response.success):
    for result in execute_response.output_parameters:
        print("{0}: {1}".format(result.name,result.value))
else:
    print (execute_response.error_message)

#Create a snapshot of the current session
response = client.create_snapshot(session_id, deployrclient.models.CreateSnapshotRequest("Iris Snapshot"),
headers)
#Return the snapshot ID for reference when you publish later.
response.snapshot_id
#If you forget the ID, list every snapshot to get the ID again.
for snapshot in client.list_snapshots(headers):
    print(snapshot)

##################################################
##        PUBLISH AS A SERVICE IN PYTHON        ##
##################################################

#Define a web service that determines the iris species by scoring
#a vector of sepal length and width, petal length and width

#Set `flower_data` for the sepal and petal length and width
flower_data = deployrclient.models.ParameterDefinition(name = "flower_data", type = "vector")
#Set `iris_species` for the species of iris
iris_species = deployrclient.models.ParameterDefinition(name = "iris_species", type = "vector")

#Define the publish request for the web service and its arguments.
#Specify the code, inputs, outputs, and snapshot.
#Don't forget to set runtime_type="Python"`.
publish_request = deployrclient.models.PublishWebServiceRequest(
       code = "iris_species = [names[x] for x in clf.predict(flower_data)]",
       input_parameter_definitions = [flower_data],
       output_parameter_definitions = [iris_species],
       runtime_type = "Python",
       snapshot_id = response.snapshot_id)

#Publish the service using the specified name (iris), version (V1.0)
client.publish_web_service_version("Iris", "V1.0", publish_request, headers)

##################################################
##           CONSUME SERVICE IN PYTHON          ##
##################################################

# Inspect holdings and metadata for service Iris V1.0.
for service in client.get_web_service_version("Iris","V1.0", headers):
    #print service metadata (description, who published,...)
    print(service)
    print(service)
    #Print input and output parameters as defined in service definition object
    print("Input Parameters: {0}".format([str(parameter) for parameter in
service.input_parameter_definitions]))
    print("Output Parameters: {0}".format([str(parameter) for parameter in
service.output_parameter_definitions]))

#Import the requests library to make requests on the server
import requests
#Import the JSON library to use for pretty printing of json responses
import json

#Create a requests `Session` object.
s = requests.Session()

#Record the R Server endpoint URL hosting the web services you created before
url = "http://localhost:12800"

#Give the request.Session object the authentication headers
#so you don't have to repeat it with each request.
s.headers = headers

#Retrieve the service-specific swagger file using the requests library.
swagger_resp = s.get(url+"/api/Iris/V1.0/swagger.json")

#Either, download service-specific swagger file using the json library.
with open('iris_swagger.json','w') as f:
   (json.dump(client.get_web_service_swagger("Iris","V1.0",headers),f, indent = 1))

#Or, print just what you need from the Swagger file,
#such as the routing paths for the endpoints to be consumed.
print(json.dumps(swagger_resp.json()["paths"], indent = 1, sort_keys = True))

#Or, print input and output parameters as defined in the Swagger.io format
print("Input")
print(json.dumps(swagger_resp.json()["definitions"]["InputParameters"], indent = 1, sort_keys = True))
print("Output")
print(json.dumps(swagger_resp.json()["definitions"]["WebServiceResult"], indent = 1, sort_keys = True))

#Make the request to consume the service using these flower_data inputs
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[7,3.2,4.7,1.4]})
#Print the output
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

#Use input from another Iris species.
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[3,2.6,3,2.5]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

#Use input from another Iris species.
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[5.1,3.5,1.4,.2]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

##################################################
##          MANAGE SERVICES IN PYTHON           ##
##################################################

#Define what needs to be updated. Here we add a description.
#Be sure to specify the runtime_type again.
update_request = deployrclient.models.PublishWebServiceRequest(
    description = "Determines iris species using length and width of sepal and petal", runtime_type =
"Python")
#Now update it by specifying the service name and version number
client.patch_web_service_version("Iris", "V1.0", update_request, headers)

#Or, publish another version of the web service, but this time
#the service returns the species as a string instead of list of strings.
flower_data = deployrclient.models.ParameterDefinition(name = "flower_data", type = "vector")
iris_species = deployrclient.models.ParameterDefinition(name = "iris_species", type = "string")
```

```
#Define the publish request for the service and its arguments.
#Specify the changed code, inputs, outputs, and snapshot.
#Don't forget to set runtime_type="Python"`.
publish_request = deployrclient.models.PublishWebServiceRequest(
    code = "iris_species = [names[x] for x in clf.predict(flower_data)][0]",
    description = "Determines the species of iris, based on Sepal Length, Sepal Width, Petal Length and Petal
Width",
    input_parameter_definitions = [flower_data],
    output_parameter_definitions = [iris_species],
    runtime_type = "Python",
    snapshot_id = response.snapshot_id)

#Now, publish the service with version (V2.0)
client.publish_web_service_version("Iris", "V2.0", publish_request, headers)

#Make request to consume service using these flower_data inputs; print output
resp = s.post(url+"/api/Iris/V2.0",json={"flower_data":[5.1,3.5,1.4,.2]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

#Return the list of all existing web services.
for service in client.get_all_web_services(headers):
    #print the service information
    print(service)
    #Print each input and output parameter
    print("Input Parameters: {0}".format([str(parameter) for parameter in
service.input_parameter_definitions]))
    print("Output Parameters: {0}".format([str(parameter) for parameter in
service.output_parameter_definitions]))

#Delete the second version we just published.
client.delete_web_service_version("Iris","V2.0",headers)
```

Walkthrough

This section describes how the code works in more detail.

Step 1. Create prerequisite client libraries

Before you can start publishing your Python code and models thorugh Microsoft Machine Learning Server, you must generate a client library using the Swagger document provided for this release.

1. Install a Swagger code generator on your local machine and familiarize yourself with it. You will use it to generate the API client libraries in Python. Popular tools include Azure AutoRest (requires Node.js) and Swagger Codegen.

2. Download the Swagger file containing the core APIs for your version of Machine Learning Server. This file contains a Swagger template defining the list of REST resources and the operations that can be called on those resources. You can find this file under `https://microsoft.github.io/deployr-api-docs/swagger/<version>/rserver-swagger-<version>.json`, where `<version>` is the 3-digit R Server version number.

   For example, for R Server 9.1 you would download from: https://microsoft.github.io/deployr-api-docs/9.1.0/swagger/rserver-swagger-9.1.0.json

3. Generate the statically-generated client library by passing the `rserver-swagger-<version>.json` file to the Swagger code generator and specifying the language you want. In this case, you should specify Python.

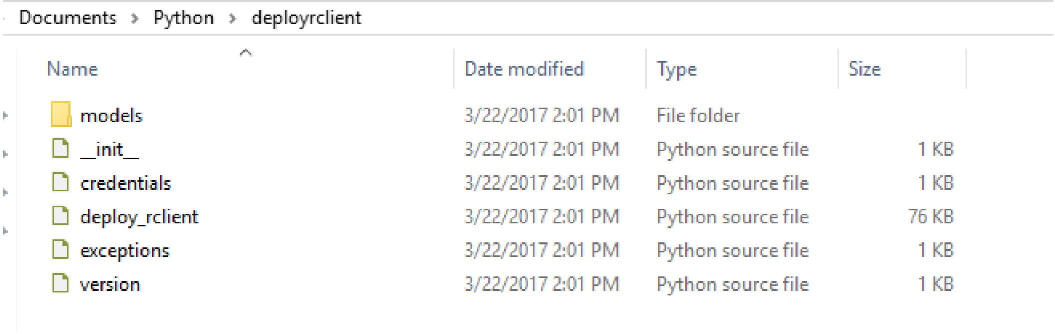
   For example, if you use AutoRest to generate a Python client library, it might look like this, where the 3-digit number represents the R Server version number:

   ```
   AutoRest.exe -Input rserver-swagger-9.1.0.json -CodeGenerator Python -OutputDirectory C:\Users\rserver-
   user\Documents\Python
   ```

You can now provide some custom headers and make other changes before using the generated client library stub. See the Command Line Interface documentation on GitHub for details regarding different configuration options and preferences, such as renaming the namespace.

1. Explore the core client library to see the various API calls you can make.

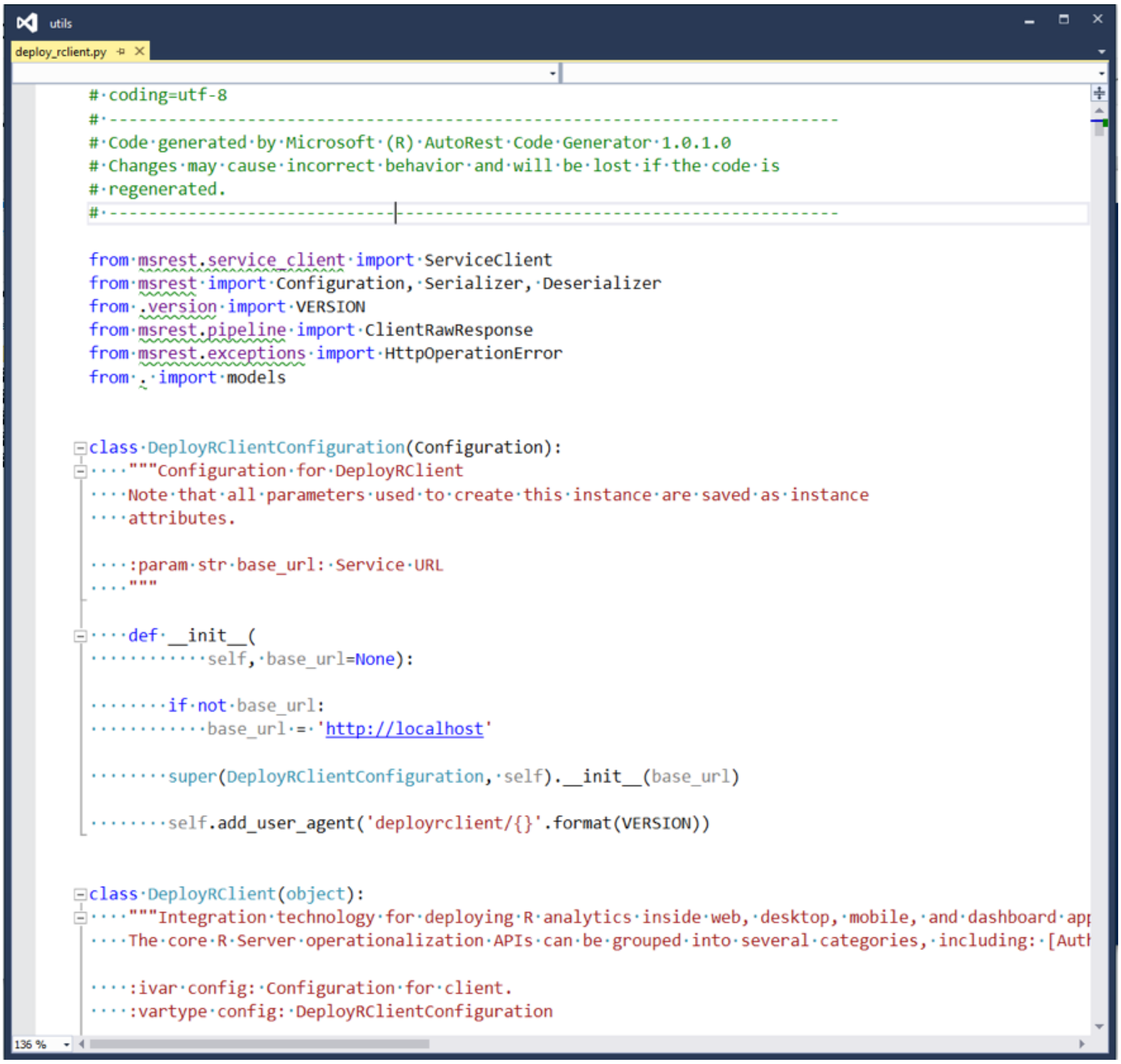
   In our example, Autorest generated some directories and files for the Python client library on your local system. By default, the namespace (and directory) is `deployrclient` and might look like this:



Documents › Python › deployrclient

Name	Date modified	Type	Size
models	3/22/2017 2:01 PM	File folder	
__init__	3/22/2017 2:01 PM	Python source file	1 KB
credentials	3/22/2017 2:01 PM	Python source file	1 KB
deploy_rclient	3/22/2017 2:01 PM	Python source file	76 KB
exceptions	3/22/2017 2:01 PM	Python source file	1 KB
version	3/22/2017 2:01 PM	Python source file	1 KB

   For this default namespace, the client library itself is called `deploy_rclient.py`. If you open this file in your IDE such as Visual Studio, you will see something like this:



```
# coding=utf-8
# --------------------------------------------------------------------------
# Code generated by Microsoft (R) AutoRest Code Generator 1.0.1.0
# Changes may cause incorrect behavior and will be lost if the code is
# regenerated.
# --------------------------------------------------------------------------

from msrest.service_client import ServiceClient
from msrest import Configuration, Serializer, Deserializer
from .version import VERSION
from msrest.pipeline import ClientRawResponse
from msrest.exceptions import HttpOperationError
from . import models


class DeployRClientConfiguration(Configuration):
    """Configuration for DeployRClient
    Note that all parameters used to create this instance are saved as instance
    attributes.

    :param str base_url: Service URL
    """

    def __init__(
            self, base_url=None):

        if not base_url:
            base_url = 'http://localhost'

        super(DeployRClientConfiguration, self).__init__(base_url)

        self.add_user_agent('deployrclient/{}'.format(VERSION))


class DeployRClient(object):
    """Integration technology for deploying R analytics inside web, desktop, mobile, and dashboard ap
    The core R Server operationalization APIs can be grouped into several categories, including: [Auth

    :ivar config: Configuration for client.
    :vartype config: DeployRClientConfiguration
```

Step 2. Add authentication and header logic

Keep in mind that all APIs require authentication; therefore, all users must authenticate when making an API call using the `POST /login` API or through Azure Active Directory (AAD).

To simplify this process, bearer access tokens are issued so that users need not provide their credentials for every single call. This bearer token is a lightweight security token that grants the “bearer” access to a protected resource: in this case, the Machine Learning Server's APIs. After a user has been authenticated, the application must validate the user’s bearer token to ensure that authentication was successful for the intended parties. To learn more about managing these tokens, see Security Access Tokens.

Before you interact with the core APIs, first authenticate, get the bearer access token using the authentication method configured by your administrator, and then include it in each header for each subsequent request:

1. Get started by importing the client library to make it accessible from your preferred Python code editor, such as Jupyter, Visual Studio, VS Code, or iPython.

   Specify the parent directory of the client library. In our example, the Autorest generated client library is under `C:\Users\rserver-user\Documents\Python\deployrclient`:

   ```
   # Import the generated client library
   import deployrclient
   ```

2. Add the authentication logic to your application to define a connection from your local machine to Machine Learning Server, provide credentials, capture the access token, add that token to the header, and use that header for all subsequent requests. Use the authentication method defined by your administrator: basic admin account, Active Directory/LDAP (AD/LDAP), or Azure Active Directory (AAD).

   AD/LDAP or `admin` account authentication

   You must call the `POST /login` API in order to authenticate. You'll need to pass in the `username` and `password` for the local administrator, or if Active Directory is enabled, pass the LDAP account information. In turn, Machine Learning Server will issue you a bearer/access token. After authenticated, the user will not need to provide credentials again as long as the token is still valid, and a header is submitted with every request. If you do not know your connection settings, please contact your administrator.

   ```
   #Using client library generated from Autorest
   #Create client instance and point it at an R Server.
   #In this case, R Server is local.
   client = deployrclient.DeployRClient("http://localhost:12800")

   #Define the login request and provide credentials.
   #Update values with the connection parameters from your admin.
   login_request = deployrclient.models.LoginRequest("<<your-username>>","<<your-password>>")
   #Make a call to the /login API.
   #Store the returned an access token in a variable.
   token_response = client.login(login_request)
   ```

   Azure Active Directory (AAD) authentication

   You must pass the AAD credentials, authority, and client ID. In turn, AAD will issue the Bearer access token. After authenticated, the user will not need to provide credentials again as long as the token is still valid, and a header is submitted with every request. If you do not know your connection settings, please contact your administrator.

```
#Import the AAD authentication library
import adal

#Define the login request and provide credentials.
#Use the AAD connection parameters provided by your admin.
url = "http://localhost:12800"
authuri = https://login.windows.net,
tenantid = "<<AAD_DOMAIN>>",
clientid = "<<NATIVE_APP_CLIENT_ID>>",
resource = "<<WEB_APP_CLIENT_ID>>",

#Acquire authentication token using AAD Device Code Login
context = adal.AuthenticationContext(authuri+'/'+tenantid, api_version=None)
code = context.acquire_user_code(resource, clientid)
print(code['message'])
token = context.acquire_token_with_device_code(resource, code, clientid)
#The authentication code returned must be entered at https://aka.ms/devicelogin
```

3. Add the `Bearer` access token and check that Machine Learning Server is currently running. This token was returned during authentication and **must be included in every subsequent request header**. This example uses a client library generated by Autorest.

> **IMPORTANT**
>
> Every API call must be authenticated. Therefore, remember to include headers with tokens to every single request.

```
#Add the returned access token to the headers variable.
headers = {"Authorization": "Bearer {0}".format(token_response.access_token)}

#Verify that the server is running.
#Remember to include `headers` in every request!
status_response = client.status(headers)
print(status_response.status_code)
```

Step 3. Prepare session and code

After authentication, you can start a Python session and create a model you'll publish later. You can include any Python code or models in a web service. Once you've set up your session environment, you can even save it as a snapshot so you can reload your session as you had it before.

> **IMPORTANT**
>
> Remember to include `headers` in every request.

1. Create a Python session on R Server. You must specify a name and the Python language (`runtime_type="Python"`). If you don't set the runtime type to Python, it defaults to R.

   This is a continuation of the example using the client library generated by Autorest:

```
#Since already logged in, create a Python session.
#Define session using name (`Session 1`) and `runtime_type="Python"`.
#Remember to specify the Python runtime type.
create_session_request = deployrclient.models.CreateSessionRequest("Session 1", runtime_type="Python")

#Make the call to start the session.
#Remember to include headers in every method call to the server.
#Returns a session ID.
response = client.create_session(create_session_request, headers)

#Store the session ID in a variable called `session_id`
#to be able to identify it later at execution time.
session_id = response.session_id
```

2. Create or call a model in Python that you'll publish as a web service

   In this example, we train a SciKit-Learn support vector machines (SVM) model on the Iris Dataset on a remote instance of Machine Learning Server. This dataset is available from the scikit-learn site.

```
#Import SVM and datasets from the SciKit-Learn library
execute_request = deployrclient.models.ExecuteRequest("from sklearn import svm\nfrom sklearn import
datasets")
execute_response = client.execute_code(session_id,execute_request, headers)
#Report if it was a success
execute_response.success

#Define the untrained Support Vector Classifier (SVC) object
#and the dataset to be preloaded.
execute_request = deployrclient.models.ExecuteRequest("clf=svm.SVC()\niris=datasets.load_iris()\nnames=
{0:'I. setosa',1:'I. versicolor',2:'I. virginica'}")
#Now, go create the object and preload Iris Dataset in R Server
execute_response = client.execute_code(session_id,execute_request, headers)
#Report if it was a success
execute_response.success

#Define two rows from the Iris Dataset as a sample for scoring
workspace_object = deployrclient.models.WorkspaceObject("species_1",[7,3.2,4.7,1.4])
workspace_object_2 = deployrclient.models.WorkspaceObject("species_2",[3,2.6,3,2.5])

#Define how to train the classifier model; what to predict; what to return
execute_request = deployrclient.models.ExecuteRequest(
    "clf.fit(iris.data, iris.target)\n"+
    "result=clf.predict(species_1)\n"+
    "other_result=clf.predict(species_2)",
    [workspace_object,workspace_object_2], #Input
    ["result", "other_result"]) #Output

#Now, go train that model on the Iris Dataset in R Server
execute_response = client.execute_code(session_id,execute_request, headers)

#If successful, print name and result of each output parameter. Else, print error.
if(execute_response.success):
    for result in execute_response.output_parameters:
        print("{0}: {1}".format(result.name,result.value))
else:
    print (execute_response.error_message)
```

3. Create a snapshot of this Python session so this environment can be saved in the web service and reproduced at consume time. Snapshots are very useful when you need a prepared environment that includes certain libraries, objects, models, files and artifacts. Snapshots save the whole workspace and working directory. However, when publishing, you can use only snapshots that you've created.

> **NOTE**
>
> While snapshots can also be used when publishing a web service for environment dependencies, it may have an impact on the performance of the consumption time. For optimal performance, consider the size of the snapshot carefully and ensure that you keep only those workspace objects you need and purge the rest. In a session, you can use the Python `del` function or the `deleteWorkspaceObject` API request to remove unnecessary objects.

```
#Create a snapshot of the current session.
response = client.create_snapshot(session_id, deployrclient.models.CreateSnapshotRequest("Iris
Snapshot"), headers)

#Return the snapshot ID for reference when you publish later.
response.snapshot_id

#If you forget the ID, list every snapshot to get the ID again.
for snapshot in client.list_snapshots(headers):
    print(snapshot)
```

Step 4. Publish the model

After your client library has been generated and you've built the authentication logic into your application, you can interact with the core APIs to create a Python session, create a model, and then publish a web service using that model.

> **NOTE**
>
> Remember that you must be authenticated before you make any API calls. Therefore, include `headers` in every request.

- Publish this SVM model as a Python web service in Machine Learning Server. This web service will score a vector that gets passed to it.

> **IMPORTANT**
>
> To ensure that the web service is registered as a Python service, be sure to specify `runtime_type="Python"`. If you don't set the runtime type to Python, it defaults to R.

```
#Define a web service that determines the iris species by scoring
#a vector of sepal length and width, petal length and width

#Set `flower_data` for the sepal and petal length and width
flower_data = deployrclient.models.ParameterDefinition(name = "flower_data", type = "vector")
#Set `iris_species` for the species of iris
iris_species = deployrclient.models.ParameterDefinition(name = "iris_species", type = "vector")

#Define the publish request for the web service and its arguments.
#Specify the code, inputs, outputs, and snapshot.
#Don't forget to set runtime_type="Python"`.
publish_request = deployrclient.models.PublishWebServiceRequest(
    code = "iris_species = [names[x] for x in clf.predict(flower_data)]",
    input_parameter_definitions = [flower_data],
    output_parameter_definitions = [iris_species],
    runtime_type = "Python",
    snapshot_id = response.snapshot_id)

#Publish the service using the specified name (iris), version (V1.0)
client.publish_web_service_version("Iris", "V1.0", publish_request, headers)
```

Step 5. Consume the web service

This section demonstrates how to consume the service in the same session where it was created.

1. In the same session, get service holdings and metadata for the service.

```
# Inspect holdings and metadata for service Iris V1.0.
for service in client.get_web_service_version("Iris","V1.0", headers):
    #print service metadata (description, who published,...)
    print(service)
    #Print input and output parameters as defined in service definition object
    print("Input Parameters: {0}".format([str(parameter) for parameter in
service.input_parameter_definitions]))
    print("Output Parameters: {0}".format([str(parameter) for parameter in
service.output_parameter_definitions]))
```

2. Examine the service holdings and prepare to consume.

```
#Import the requests library to make requests on the server
import requests
#Import the JSON library to use for pretty printing of json responses
import json

#Create a requests `Session` object.
s = requests.Session()

#Record the R Server endpoint URL hosting the web services you created
url = "http://localhost:12800"

#Give the request.Session object the authentication headers
#so you don't have to repeat it with each request.
s.headers = headers

# Retrieve the service-specific swagger file using the requests library.
swagger_resp = s.get(url+"/api/Iris/V1.0/swagger.json")

#Either download service-specific swagger file using the json library.
with open('iris_swagger.json','w') as f:
   (json.dump(client.get_web_service_swagger("Iris","V1.0",headers),f, indent = 1))

#Or print just what you need from the Swagger file,
#such as the routing paths for the endpoints to be consumed.
print(json.dumps(swagger_resp.json()["paths"], indent = 1, sort_keys = True))

#Or, print input and output parameters as defined in the Swagger.io format
print("Input")
print(json.dumps(swagger_resp.json()["definitions"]["InputParameters"], indent = 1, sort_keys = True))
print("Output")
print(json.dumps(swagger_resp.json()["definitions"]["WebServiceResult"], indent = 1, sort_keys = True))
```

3. Consume the service by supplying some input and getting the Iris species.

```
#Make the request to consume the service using these flower_data inputs
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[7,3.2,4.7,1.4]})
#Print the output
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

##Use input from another Iris species.
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[3,2.6,3,2.5]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))

##Use input from another Iris species.
resp = s.post(url+"/api/Iris/V1.0",json={"flower_data":[5.1,3.5,1.4,.2]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))
```

Managing the Services

Now that you've created a web service, you can update, delete, or republish that service. You can also list all the web services that are hosted using Microsoft Machine Learning Server.

Update a web service

You can update a web service to change the code, model, description, inputs, outputs and more. In this example, we update the service to add a description useful to people who might consume this service.

```
#Define what needs to be updated. Here we add a description.
#Be sure to specify the runtime_type again.
update_request = deployrclient.models.PublishWebServiceRequest(
    description = "Determines iris species using length and width of sepal and petal", runtime_type =
"Python")
#Now update it by specifying the service name and version number
client.patch_web_service_version("Iris", "V1.0", update_request, headers)
```

Publish another version

You can also publish another version of the web service. In this example, the service will now return the Iris species as a string instead of as a list of strings.

```
#Publish another version of the web service, but this time
#the service returns the species as a string instead of list of strings.
flower_data = deployrclient.models.ParameterDefinition(name = "flower_data", type = "vector")
iris_species = deployrclient.models.ParameterDefinition(name = "iris_species", type = "string")

#Define the publish request for the service and its arguments.
#Specify the changed code, inputs, outputs, and snapshot.
#Don't forget to set runtime_type="Python"`.
publish_request = deployrclient.models.PublishWebServiceRequest(
    code = "iris_species = [names[x] for x in clf.predict(flower_data)][0]",
    description = "Determines the species of iris, based on Sepal Length, Sepal Width, Petal Length and Petal
Width",
    input_parameter_definitions = [flower_data],
    output_parameter_definitions = [iris_species],
    runtime_type = "Python",
    snapshot_id = response.snapshot_id)

#Now, publish the service with version (V2.0)
client.publish_web_service_version("Iris", "V2.0", publish_request, headers)

#Make request to consume service using these flower_data inputs; print output
resp = s.post(url+"/api/Iris/V2.0",json={"flower_data":[5.1,3.5,1.4,.2]})
print(json.dumps(resp.json(), indent = 1, sort_keys = True))
```

List services

Get a list of all web services, including those created by other users or in different languages.

```
#Return the list of all existing web services.
for service in client.get_all_web_services(headers):
    #print the service information
    print(service)
    #Print each input and output parameter
    print("Input Parameters: {0}".format([str(parameter) for parameter in
service.input_parameter_definitions]))
    print("Output Parameters: {0}".format([str(parameter) for parameter in
service.output_parameter_definitions]))
```

Delete services

You can delete services you've created. You can also delete the services of others if you are assigned to a role that has the appropriate permissions.

In this example, we delete the second web service version we just published.

```
#Delete the second version we just published.
client.delete_web_service_version("Iris","V2.0",headers)
```

Using R Code in Transact-SQL (SQL Server R Services)

4/29/2017 • 2 min to read • Edit Online

This tutorial walks you through the basic mechanics of calling an R script from a T-SQL stored procedure.

What You'll Learn

- About R and SQL data types and data objects
- How to embed R in a T-SQL function
- Creating a simple model and saving it in SQL Server
- Creating predictions and an R plot using the model

Estimated Time

30 minutes, not including setup

Prerequisites

You must have access to an instance of SQL Server where R Services is already installed. The instance can be in an Azure virtual machine or on-premises. YOu can use either SQL Server 2016 or SQL Server 2017.

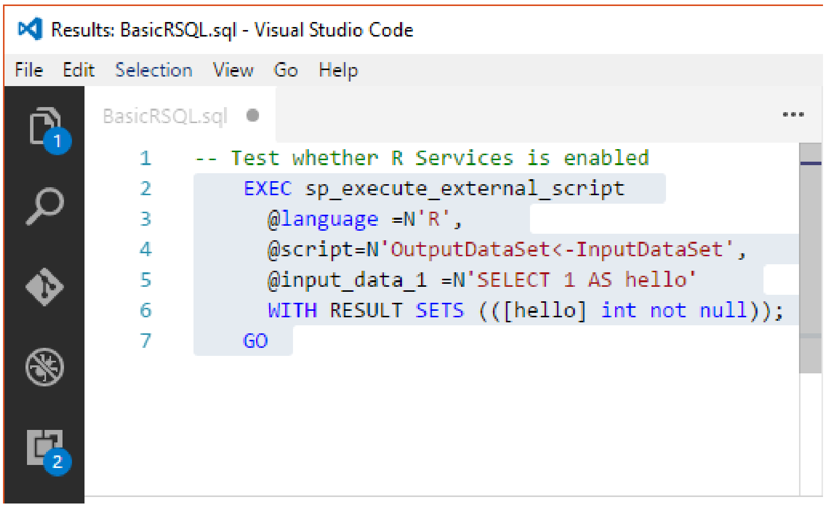
To run SQL queries that include R script, use SQL Server Management Studio (SSMS), Visual Studio, or any other application that can connect to a database and run ad hoc T-SQL code. To demonstrate how easy it is to run R inside SQL Server, we'll use the new **mssql extension for Visual Studio Code**, a free development environment that can run on Linux, macOS, or Windows. To install it, 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 will prompt you to create a profile to use when connecting to a specific database. This is optional, but will make it easier to switch between databases and logins.
   - Choose a server or instance where R Services has already 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.

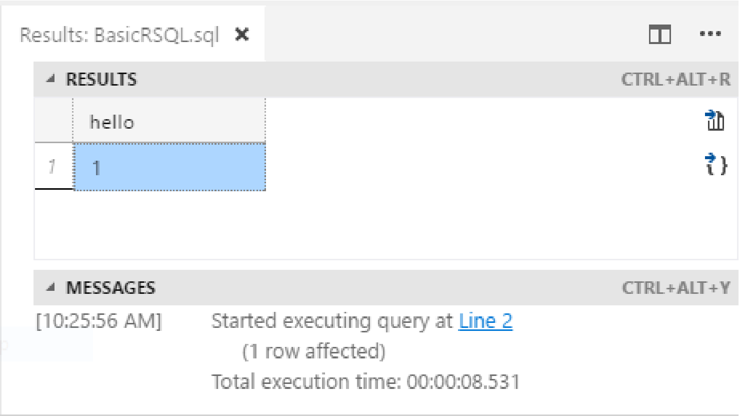
   ```
   EXEC sp_execute_external_script
     @language =N'R',
     @script=N'OutputDataSet<-InputDataSet',
     @input_data_1 =N'SELECT 1 AS hello'
     WITH RESULT SETS (([hello] int not null));
   GO
   ```

   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.


Results: BasicRSQL.sql - Visual Studio Code
File Edit Selection View Go Help
BasicRSQL.sql

```
-- Test whether R Services is enabled
    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
```

Results



	hello
1	1

```
[10:25:56 AM]     Started executing query at Line 2
                      (1 row affected)
                  Total execution time: 00:00:08.531
```

Troubleshooting

- If you get any errors from this query, installation of R Services might be incomplete. After adding the feature using the SQL Server setup wizard, you must take some additional steps to enable use of external code libraries. See Set up SQL Server R Services.

- 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. See R Services Installation and Upgrade FAQ

- To get Visual Studio Code, see Download and install Visual Studio Code.

Next Step

Now that your instance of R Services is ready, let's get started.

Step 1: Working with Inputs and Outputs

Step 2: R and SQL Data Types and Data Objects

Step 3: Using R Functions with SQL Server Data

Step 4: Create a Predictive Model

Step 5: Predict and Plot from Model

Working with Inputs and Outputs (R in T-SQL Tutorial)

4/19/2017 • 4 min to read • Edit Online

When you want to run R code in SQL Server, you must wrap the R script in a system stored procedure, sp_execute_external_script. This 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.

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

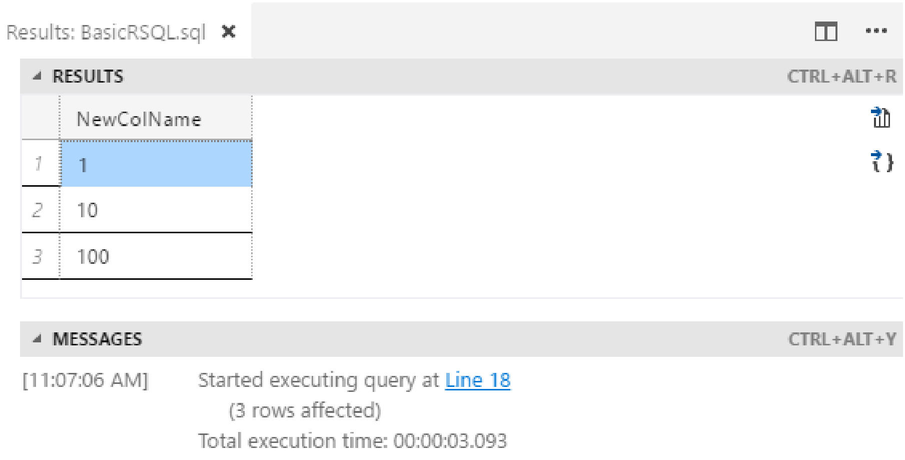
COL1
1
10
100

Get the same data using R script

After the table has been created, run the following statement. It gets the data from the table, makes a round-trip through the R runtime, and returns the values with the column name, *NewColName*.

```
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));
```

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
```

Notes

- 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 *InputDatSet*, 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 this error?

Error in eval(expr, envir, enclos) : 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 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.

This is a generic error that you'll see often while testing your R code. It means that the R script ran successfully, but SQL Server got back no data, or got back 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.

- 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.
- In Management Studio, tabular results are returned in the **Values** pane. Messages returned by the R runtime are provided in the **Messages** pane.

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

Col1
hello
□
world

Next Step

You'll 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.

R and SQL Data Types and Data Objects

R and SQL Data Types and Data Objects (R in T-SQL Tutorial)

4/19/2017 • 8 min to read • Edit Online

In this step, you'll learn about some common issues that arise when moving data between R and SQL Server.

- Data types sometimes do not match
- Implicit conversions are performed
- Cast and convert operations are sometimes required
- R and SQL use different data objects

Always return R data as 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'll 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 R scripts, which look almost identical. The first returns a single column of three values, and 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'  '
      ;
```

Identifying the schema and data types of R data

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");
      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 on 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 (`is.matrix`, `is.vector`, etc.).

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 heterogenous data types.

For example, assume you run the following statement to perform matrix multiplication using R. You multiply a single-column matrix with the 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(InputDataSet);
        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 if you change the size of the array `y`.

```
execute sp_execute_external_script
   @language = N'R'
   , @script = N'
        x <- as.matrix(InputDataSet);
        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 will return 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!

Merge or multiply columns of different length

R provides a lot of 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 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.

TIP

See this article for more help on navigating R data frames: 15 Easy Solutions to Your Data Frame Problems in R

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.
- 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.

```
SELECT ReportingDate
         , CAST(ModelRegion as varchar(50)) as ProductSeries
         , Amount
           FROM [AdventureWorksDW2014].[dbo].[vTimeSeries]
           WHERE [ModelRegion] = 'M200 Europe'
           ORDER BY ReportingDate ASC
```

NOTE

Any version of AdventureWorks will do, or you can use a query of your own. The point is to try to handle some data containing text, datetime and numeric values.

Now, try pasting this query into an R script wrapper. 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.

```
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 [AdventureWorksDW2014].[dbo].[vTimeSeries]
           WHERE [ModelRegion] = ''M200 Europe''
           ORDER BY ReportingDate ASC ;'
WITH RESULT SETS undefined;
```

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

From this, you can see a couple of changes in even a small query:

- 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

Some SQL Server data types are not supported by R. 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.

For more information on supported and unsupported data types, see Working with R 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 step, you'll learn how to apply R functions to SQL Server data.

Using R Functions with SQL Server Data

Using R Functions with SQL Server Data (R in T-SQL Tutorial)

4/19/2017 • 3 min to read • Edit Online

Now that you're familiar with basic operations, it's time to have some fun with R. For example, many advanced statistical functions might be complicated to implement using T-SQL, but require only a single line of R code. With R Services, it's easy to embed R utility scripts in a stored procedure.

In these examples, you'll embed R mathematical and utility functions in a SQL Server stored procedure.

Create a stored procedure to generate random numbers

For simplicity, we'll use the R `stats` package, which is installed and loaded by default with R Services. The package contains hundreds of functions for common statistical tasks, among them the `rnorm` function, which generates a specified number of random numbers using the normal distribution, given a standard deviation and mean.

For example, this R code returns 100 numbers on a mean of 50, given a standard deviation of 3.

```
as.data.frame(rnorm(100, mean = 50, sd = 3));
```

To call this line of R from T-SQL, run sp_execute_external_script and add the R function in the R script parameter, like this:

```
EXEC sp_execute_external_script
      @language = N'R'
    , @script = N'
         OutputDataSet <- as.data.frame(rnorm(100, mean = 50, sd =3));'
    , @input_data_1 = N'   ;'
      WITH RESULT SETS (([Density] float NOT NULL));
```

What if you'd like to make it easier to generate a different set of random numbers?

Easy: define a stored procedure that gets the arguments from the user. Then, pass those arguments into the R script as variables.

```
CREATE PROCEDURE MyRNorm (@param1 int, @param2 int, @param3 int)
AS
    EXEC sp_execute_external_script
      @language = N'R'
    , @script = N'
         OutputDataSet <- as.data.frame(rnorm(mynumbers, mymean, mysd));'
    , @input_data_1 = N'   ;'
    , @params = N' @mynumbers int, @mymean int, @mysd int'
    , @mynumbers = @param1
    , @mymean = @param2
    , @mysd = @param3
    WITH RESULT SETS (([Density] float NOT NULL));
```

- The first line defines each of the SQL input parameters that are required when the stored procedure is executed.
- The line beginning with `@params` defines all variables used by the R code, and the corresponding SQL data types.

- The lines that immediately follow map the SQL parameter names to the corresponding R variable names.

Now that you've wrapped the R function in a stored procedure, you can easily call the function and pass in different values, like this:

```
EXEC MyRNorm @param1 = 100,@param2 = 50, @param3 = 3
```

Notes

- Want to install more R packages, to get more advanced statistical functions? See Installing and Managing R packages.
- The R Services team created an R package to help you convert your standalone R code to a format that can be easily parameterized using SQL Server stored procedures. For more information, see How to Create a Stored Procedure using sqlrutils.

Use R utility functions for troubleshooting

By default, R Services includes the `utils` package, which provides a variety of utility functions for investigating the current R environment. This can be useful if you are finding discrepancies in the way your R code performs in SQL Server and in outside environments.

For example, you might use the R `memory.limit()` function to get memory for the current R environment. Because the `utils` package is installed but not loaded by default, you must use the `library()` function to load it first.

```
EXECUTE sp_execute_external_script
      @language = N'R'
    , @script = N'
        library(utils);
        mymemory <- memory.limit();
        OutputDataSet <- as.data.frame(mymemory);'
    , @input_data_1 = N' ;'
WITH RESULT SETS (([Col1] int not null));
```

You might also use the system timing functions in R, such as `system.time` and `proc.time`, to capture the time used by R processes and analyze performance issues.

For an example, see this tutorial: Create Data Features. In this walkthrough, R timing functions are embedded in the solution to compare the performance of two methods for creating features from data: R functions vs. T-SQL functions.

Next Step

Next, you'll build a predictive model using R in SQL Server.

Create a Predictive Model

Create a Predictive Model (R in T-SQL Tutorial)

4/19/2017 • 4 min to read • Edit Online

In this step, 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 already included with R, because it is small and easy to understand.

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'
```

- If you want to use temporary tables, be aware that some R clients will 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

If you need a refresher on linear models, see this tutorial, which describes the process of fitting a linear models 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'lrmodel <- rxLinMod(formula = distance ~ speed, data = CarsData);
        trained_model <- data.frame(payload = as.raw(serialize(lrmodel, 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

Now you'll 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 a second time, you'll 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.)

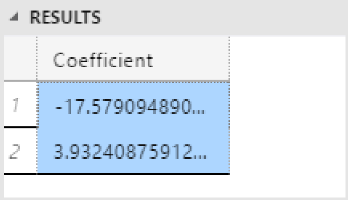
However, you can return outputs of other types, such as scalars, in addition to the data frame.

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 callings variable, or you could save it 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



	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 of sp_execute_external_script.
- 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 Step

Now that you have a model, in the final step, you'll learn how to generate predictions from it and plot the results.

Predict and Plot from Model

Predict and Plot from Model (R in T-SQL Tutorial)

4/19/2017 • 6 min to read • Edit Online

To score new data, you'll get one of the trained models from the table, and then call a new set of data on which to base predictions.

Create the table of new speeds

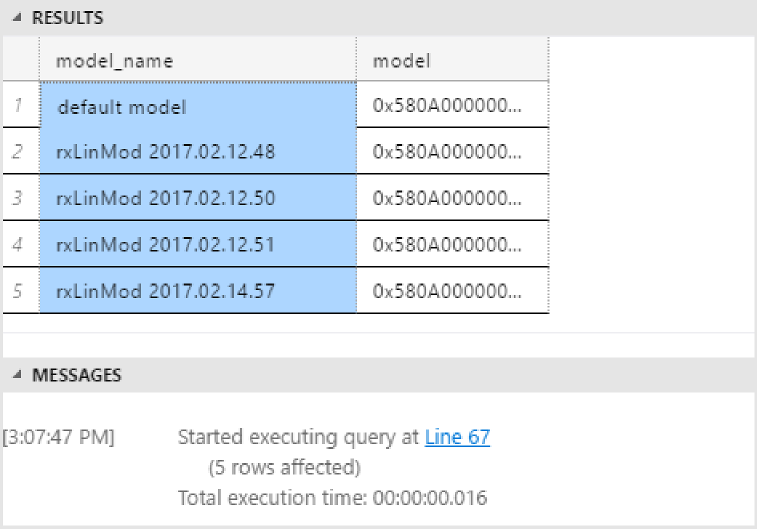
Did you notice that the original training data stops at a speed of 25 miles per hour? That's because the original data was based on an experiment from 1920!

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 will 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	model
1	default model	0x580A000000...
2	rxLinMod 2017.02.12.48	0x580A000000...
3	rxLinMod 2017.02.12.50	0x580A000000...
4	rxLinMod 2017.02.12.51	0x580A000000...
5	rxLinMod 2017.02.14.57	0x580A000000...

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 a 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

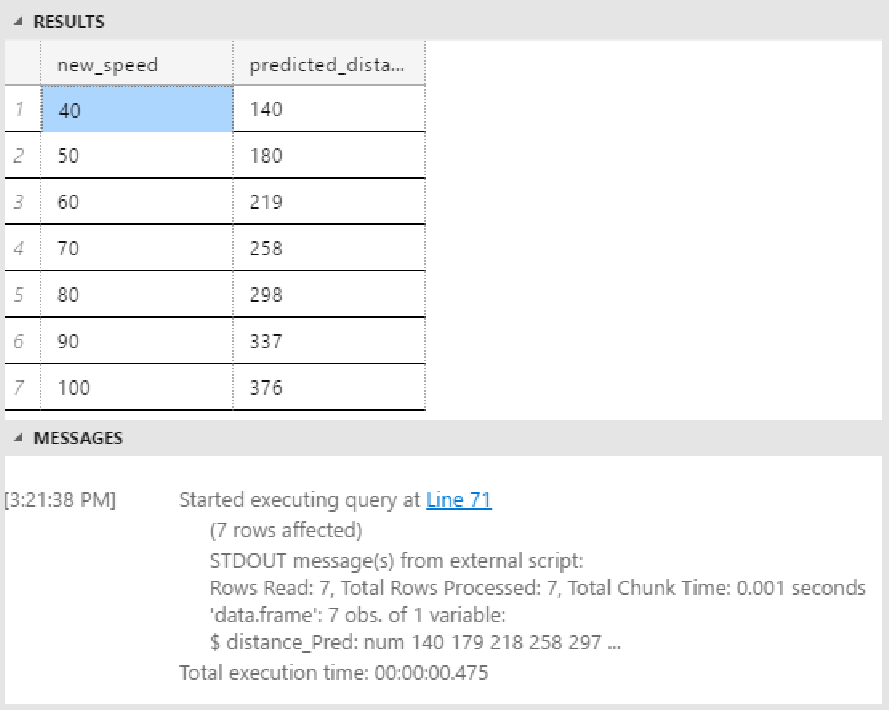
In this example, because your model is based on the **rxLinMod** algorithm provided as part of the **RevoScaleR** package, you should call the `rxPredict` function, rather than the generic R `predict` function.

```
DECLARE @speedmodel varbinary(max) = (SELECT model FROM [dbo].[stopping_distance_models] WHERE model_name =
'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);
            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))
```

Notes

- 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.
- Apply the `rxPredict` function with appropriate arguments to the model, and provide the new input data.
- We used the `str` function while testing to check the schema of data being returned from R. You can always remove the statement later.
- You can add column names to the output data frame as part of your R script, but here we just used the WITH RESULTS clause.
- To return columns from the original dataset together with the prediction results, concatenate the source column with the predicted values column as part of your R script, and then return the data frame to SQL Server.

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

```
[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 are parallelizable. 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* parameter.

```
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))
```

Notes

- Parallel execution provides benefits only when working with very large data. 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.

Note that 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.

Therefore, to get the benefits of parallel processing when you train your model, we recommend that you use one of the **ScaleR** 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 ScaleR, see ScaleR Distributed Computing.

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 in R Services 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 an application that can display images, such as Reporting Services.

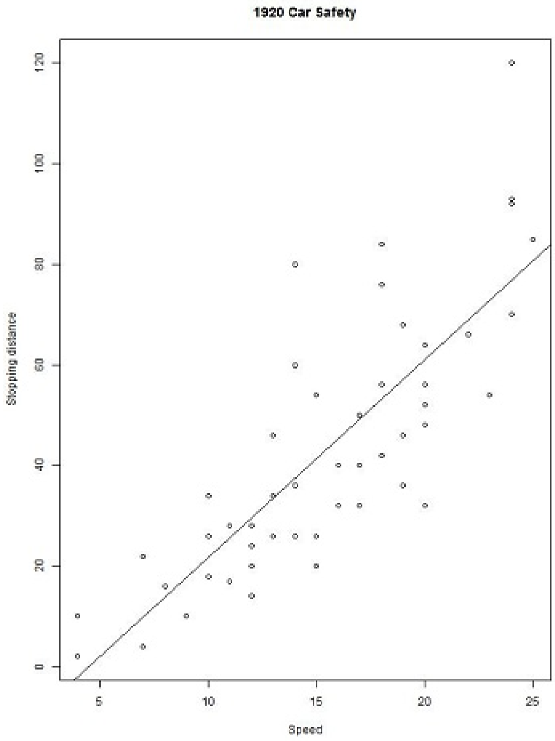
The following example demonstrates how to create a simple graphic using a plotting function included by default with R. The image is output to the specified file, and is also output into a SQL variable by the stored procedure.

```
 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)));
```

Notes

- The `tempfile` function returns a string that can be used as a file name, but the file is not actually generated yet.
- For arguments to `tempfile`, you can specify a prefix and file extension, as well as a tmpdir. To verify the file name and path, we printed a message using `str()`.
- The `jpeg` function creates an R device with the specified parameters.
- After you have created the plot, you can add more visual features to it. In this case, a regression line was 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 containing 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. Also requires the **reshape** package.

- Create Graphs and Plots Using R: Lesson 2 in an end-to-end solution, based on the NYC taxi data.

Conclusion

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.

To continue learning about solutions using R with SQL Server, see the tutorials and end-to-end scenarios created by the Microsoft Data Science and R Services development teams:

- SQL Server R Services tutorials

For guidance on using the new RevoScaleR packages, see these resources for Microsoft R:

- Explore R and ScaleR in 25 functions
- Fitting Linear Models
- Models in RevoScaleR

Data Science End-to-End Walkthrough

4/29/2017 • 3 min to read • Edit Online

In this walkthrough, you'll develop an end-to-end solution for predictive modeling using R Services (In-Database).

This walkthrough is based on a popular set of public data, the New York City taxi dataset. You will 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 will get a tip on a particular taxi trip. You'll also deploy your R model to SQL Server and use server data to generate scores based on the model.

This example can easily be extended to all kinds of real-life problems, such as predicting customer responses to sales campaigns, or predicting spending by visitors to events. Because the model can be invoked from a stored procedure, you can also easily embed it in an application.

Intended 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 creating queries using Transact-SQL.

- You will be given the SQL and R scripts to execute.
- No R coding is required; all scripts are provided.

Prerequisites

- You must have access to an instance of SQL Server 2016, or an evaluation version of SQL Server 2017.
- At least one instance on the SQL Server computer must have R Services (In-Database) installed.
- To run R commands, you'll need a separate computer that has an R IDE and the Microsoft R Open libraries. It can be a laptop or other networked computer, but it must be able to connect to the SQL Server instance.

For details about how to set up these server and client environments, see Prerequisites for Data Science Walkthroughs (SQL Server R Services).

What the Walkthrough Covers

Note that the estimated times do not include setup.

LESSON LIST	ESTIMATED TIME
Lesson 1: Prepare the Data (Data Science End-to-End Walkthrough) Obtain the data used for building a model. Download a public dataset and load it into a SQL Server database.	30 minutes
Lesson 2: View and Explore the Data (Data Science End-to-End Walkthrough) Explore the data, prepare it for modeling, and create new features. You'll use both SQL and R to explore the data and generate summaries.	20 minutes

LESSON LIST	ESTIMATED TIME
Lesson 3: Create Data Features (Data Science End-to-End Walkthrough) Perform feature engineering using custom functions in R and Transact-SQL. Compare the performance of R and T-SQL for featurization tasks.	10 minutes
Lesson 4: Build and Save the Model (Data Science End-to-End Walkthrough) Train and tune a predictive model. Assess model performance. This walkthrough creates a classification model. Plot the model's accuracy using R.	15 minutes
Lesson 5: Deploy and Use the Model (Data Science End-to-End Walkthrough) Deploy the model in production by saving the model to a SQL Server database. Call the model from a stored procedure to generate predictions.	10 minutes

Notes

- The walkthrough is designed to introduce R developers to R Services (In-Database), so R is used whever possible. 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'll point out possible optimizations along the way.
- The walkthrough was originally developed for and tested on SQL Server 2016. However, screenshots and procedures have been updated to use the latest version of SQL Server Management Studio, which works with SQL Server 2017.

Next Step

Lesson 1: Prepare the Data (Data Science End-to-End Walkthrough)

Prerequisites for Data Science Walkthroughs (SQL Server R Services)

4/19/2017 • 1 min to read • Edit Online

We recommend that you do the walkthroughs on an R workstation that can connect to a SQL Server computer on the same network. You can also run the walkthrough on a computer that has both SQL Server and an R development environment.

Install SQL Server 2016 R Services (In-Database)

You must have access to an instance of SQL Server 2017 with R Services (In-Database) installed. For more information about how to set up R Services (In-Database), see Set up SQL Server R Services (In-Database.

IMPORTANT

Be sure to use SQL Server 2017 or later. Previous versions of SQL Server do not support integration with R. However, you can use older SQL databases as an ODBC data source.

Install an R Development Environment

To complete this walkthrough on your computer, you will need an R development environment, or any other command line tool that can run R commands.

- **R Tools for Visual Studio** is a free plug-in that provides Intellisense, debugging, and support for Microsoft R Server and SQL Server R Services. To download, see R Tools for Visual Studio.

- **Microsoft R Client** is a lightweight development tool that supports development in R using the ScaleR packages. 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/.

  However, 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.

- R tools (R.exe, RTerm.exe, RScripts.exe) are installed by default when you install Microsoft R Open. If you do not wish to install an IDE you can use these tools.

Get Permissions to Connect to SQL Server

In this walkthrough, you will connect to an instance of SQL Server to run scripts and upload data. To do this, 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 create an account for you on the server with the following privileges on the database where you will be using R:

- Create database, tables, functions, and stored procedures
- Insert data into tables

Start the Walkthrough

Lesson 1: Prepare the Data (Data Science End-to-End Walkthrough)

4/29/2017 • 11 min to read • Edit Online

If you have a clean installation of SQL Server 2016 or 2017 with R Services, you're ready to get started. In this lesson, you'll get the data and R packages needed for the scenario.

1. Download the data and all R scripts from Github. A PowerShell script is provided for convenience.

2. Install some additional R packages, both on the server and on your R workstation.

3. Set up the database that will be used for modeling and scoring.

   For this, you'll use a second PowerShell script, RunSQL_R_Walkthrough.ps1. The script configures the database and uploads the data into the table you specify. It also creates some SQL functions and stored procedures that simplify data science tasks.

1. Download the Data and Scripts

All the code that you will has been provided in a GitHub repository. You can use a PowerShell script to make a local copy of the files.

Download scripts using PowerShell

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.

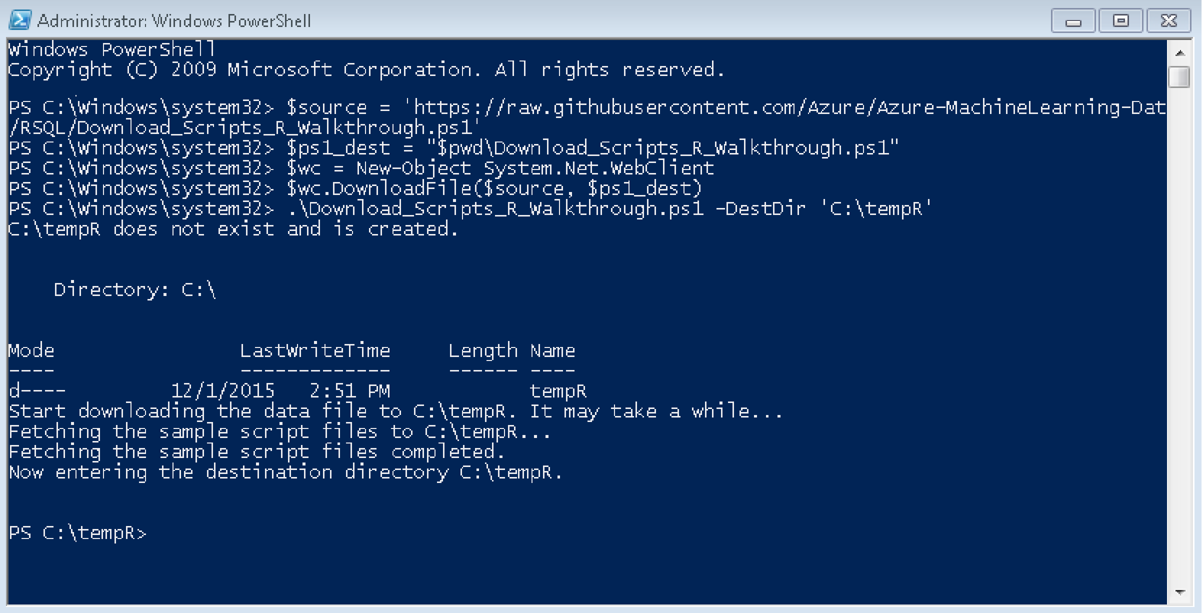
   ```
   Set-ExecutionPolicy Unrestricted -Scope Process -Force
   ```

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 will create the folder for you.

4. Downloading might take a while. After it is complete, the Windows PowerShell command console should look like this:



```
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 filesls , see What's Included.

2. Install Required 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 will be developing the solution, and on the SQL Server computer where you will 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 will also install required packages. In all, about 32 packages are required.

Install required packages on the 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).

   Do not install to a user library if prompted, because if you do, the SQL Server instance will not find the packages in the R library used by R Services. For more information, see Installing New R Packages on SQL

Server.

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])
   ```

Notes:

- This example uses the R grep function to search the vector of available paths and find the one in “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 using the R function, `installed.packages()`.

3. Prepare the environment using PowerShell script RunSQL_R_Walkthrough.ps1

When you downloaded the data files along with R and T-SQL scripts, the PowerShell script `RunSQL_R_Walkthrough.ps1` should be included in the download.

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.

You should run this script on the computer where you will be building 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.

To run the script

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 will be prompted for each of the following parameters:

   Database server name: The name of the SQL Server instance where R Services is installed.

   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 will default to using integrated Windows authentication.

> **WARNING**
>
> When you use the prompt in the PowerShell script to provide your credentials, the password will be 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.csv1`.

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, load the data manually as described in the troubleshooting section.

Results (successful completion)

```
Execution successful
C:\tempR\fnEngineerFeatures.sql execution done
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 here to jump to the next lesson!

Lesson 2: View and Explore the Data (Data Science End-to-End Walkthrough)

Troubleshooting

If you run into trouble, you can run all or any of the steps manually, using the lines of the PowerShell script as examples.

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 will need the path to load the data to the table using **bcp**.

Unable to download the data

The data file is large. 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.

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 SqlUsersPassword -
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 a table of the same name and the same schema, bcp will insert a new copy of the data rather than overwriting existing data. To avoid duplicate data, truncate any existing tables before re-running the script.

What the Download Includes

When you download the files from the GitHub repository, you'll get the following:

- Data in CSV format
- 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 learn more about how this data was originally collected, and how you can get the full data set, see http://chriswhong.com/open-data/foil_nyc_taxi/.

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 350MB.

Files

- **RunSQL_R_Walkthrough.ps1** You'll 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 will be 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.

SQL Scripts

This PowerShell script executes multiple Transact-SQL scripts on the SQL Server instance. The following table lists the Transact-SQL script files.

SQL SCRIPT FILE NAME	WHAT IT DOES
create-db-tb-upload-data.sql	Creates database and two tables: *nyctaxi_sample*: 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. *nyc_taxi_models*: An empty table that you'll use later to save the trained classification model.

SQL SCRIPT FILE NAME	WHAT IT DOES
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

Lesson 2: View and Explore the Data (Data Science End-to-End Walkthrough)

Previous Lesson

Data Science End-to-End Walkthrough

Lesson 2: View and Explore the Data (Data Science End-to-End Walkthrough)

4/19/2017 • 1 min to read • Edit Online

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'll explore the data objects and generate plots, using both Transact-SQL and R functions included in R Services (In-Database).

Then you will generate plots to visualize the data, using new functions provided by packages installed with R Services (In-Database).

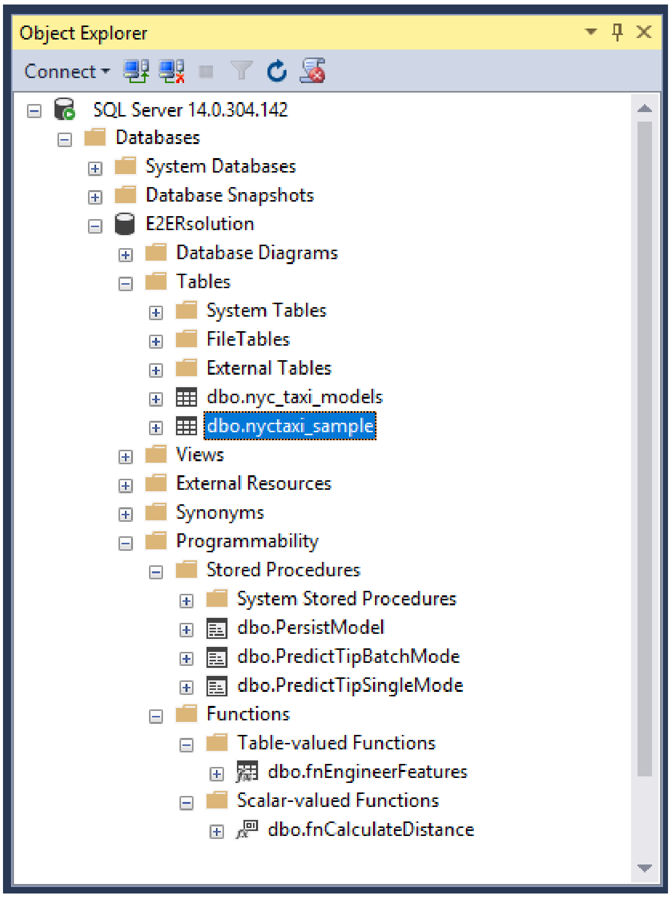
> **TIP**
>
> Already an R maestro?
>
> Now that you've downloaded all the data and prepared the environment, you are welcome to run the complete R script in RStudio or any other environment, and explore the functionality on your own. Just open the file RSQL_Walkthrough.R and highlight and run individual 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 VS Code.

2. Click on the database you created, and expand to see the new database, tables, and functions



3. To verify that the data was loaded correctly, right-click the table and select **Select top 1000 rows** to run 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. For details about how that was done, review the script used to create the table, `create-db-tb-upload-data.sql`

   For now, 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 Steps

View and Summarize Data using R (Data Science End-to-End Walkthrough)

Create Graphs and Plots Using R (Data Science End-to-End Walkthrough)

Previous Lesson

Lesson 1: Prepare the Data (Data Science End-to-End Walkthrough)

See Also

Columnstore Indexes Guide

Lesson 2-1 - View and Summarize Data using R

4/19/2017 • 7 min to read • Edit Online

Now you'll work with the same data using R code. You'll also learn how to use the functions in the **RevoScaleR** package included with R Services (In-Database).

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.

Save your R workspace in case you want to complete the rest of the walkthrough later. That way the data objects and other variables will be ready for re-use.

Define a SQL Server Compute Context

To get data from SQL Server to use in your R code, you need to:

- 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, you can define transformation 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 IDE.

1. If the **RevoScaleR** package is not already loaded, run:

   ```
   library("RevoScaleR")`.
   ```

   If you get an error, make sure that your R development environment is using the library that includes the RevoScaleR package. Use a command such as `.libPaths())` to view the current path.

2. Create the connection string for SQL Server.

   ```
   # SQL authentication
   connStr <- "Driver=SQL
   Server;Server=Your_Server_Name.somedomain.com;Database=Your_Database_Name;Uid=Your_User_Name;Pwd=Your_Pa
   ssword"
   ```

```
# Windows authentication
connStrWin <- "Driver=SQL Server;Server=SQL_instance_name;Database=database_name;Trusted_Connection=Yes"

# Map the connection string to the one used in the rest of this tutorial
connStr \<- connStrWin
```
> [!NOTE]
> The R script available for download uses SQL logins only. In this tutorial, we've provided examples of both
SQL logins and Windows integrated authentication. We recommend that you use Windows authentication where
possible, to avoid saving passwords in your R code.
>
> Regardless of which credentials you use, the account that you use must have permissions to read data and to
create new tables in the specified database. For information about how to add users to the SQL database and
give them the correct permissions, see [Post-Installation Server Configuration &#40;SQL Server R Services&#41;]
(http://msdn.microsoft.com/library/b32d43de-7e9c-4ae4-a110-d8d56b514172).
```

1. Define variables to use in a new *compute context*. By setting the compute context to the SQL Server instance, you can take advantage of server resources, rather than trying to run it in the memory on your laptop.

   ```
   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 or not. For a discussion of waiting vs. non-waiting jobs, see ScaleR Distributed Computing.

   - Use the argument *sqlConsoleOutput* to indicate that you dont want to see output from the R console.

2. Instantiate the compute context object with the variables and connection strings already defined, and save it in the R variable *sqlcc*.

   ```
   cc <- RxInSqlServer(connectionString = connStr, shareDir = sqlShareDir,
                       wait = sqlWait, consoleOutput = sqlConsoleOutput)
   ```

3. By default, the compute context is local, so you'll need to explicitly set the *active compute context*.

   ```
   rxSetComputeContext(cc)
   ```

   - `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 an RxSqlServer Data Source

A *data source* specifies some set of data that you want to use for a task, such as training, exploration, scoring, or generating features.

You already defined the database you want to work with, and saved that information in R variables. Now you can re-use that data connection to create data objects, by calling the *RxSqlServer* function.

1. Save the SQL statement as a string variable. The query defines the data you'll use to train the model.
   ```

```
sampleDataQuery <- "SELECT TOP 1000 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"
```

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 Working with R Data Types.

   - The argument *rowsPerRead* 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 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 too small a value can also provide slower performance.

3. At this point, the *inDataSource* object doesn't contain any data from the SQL query. The data is not pulled into the local environment until you run a function such as *rxImport* or *rxSummary*.

   However, this object is a convenient shortcut for defining the data. You can call the data source using multiple functions, to move data, to get a summary of the data and its variables, to manipulate and transform the data, or to use it for training a model.

Use the SQL Server Data in R

You can now apply R functions to the data source, to explore, summarize, and chart the SQL Server data. In this section, you 'll try out several of the functions provided in R Services (In-Database) that support remote compute contexts.

1. Call the function *rxGetVarInfo*, using the data source *inDataSource* as an argument, to get a list of the variables in the data source and their data types.

```
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

rxGetVarInfo can be used with any data frame, or 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.

2. Call the RevoScaleR function *rxSummary* to summarize the fare amount, based on the number of passengers.

   This function can be used to get more detailed statistics about individual variables. You can also transform values, compute summaries using factor levels, and save the summaries for re-use.

   ```
   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. The **rxSummary** function also requires that you specify minimum and maximum values for the passenger_count column: : here, 1 and 6.
   - 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

   ```
   rxSummary(formula = ~fare_amount:F(passenger_count), data = inDataSource)
   Summary Statistics Results for: ~fare_amount:F(passenger_count)
   Data: inDataSource (RxSqlServerData Data Source)
   Number of valid observations: 1000
   Name Mean StdDev Min Max ValidObs MissingObs
   fare_amount:F_passenger_count 12.4875 9.682605 2.5 64 1000 0
   Statistics by category (6 categories):
   Category F_passenger_count Means StdDev Min
   fare_amount for F(passenger_count)=1 1 12.00901 9.219458 2.5
   fare_amount for F(passenger_count)=2 2 11.61893 8.858739 3.0
   fare_amount for F(passenger_count)=3 3 14.40196 10.673340 3.5
   fare_amount for F(passenger_count)=4 4 13.69048 8.647942 4.5
   fare_amount for F(passenger_count)=5 5 19.30909 14.122969 3.5
   fare_amount for F(passenger_count)=6 6 12.00000 NA 12.0
   Max ValidObs
   55 666
   52 206
   52 51
   39 21
   64 55
   12 1
   "It takes CPU Time=0.5 seconds, Elapsed Time=4.59 seconds to summarize the inDataSource."
   ```

Next Step

Create Graphs and Plots Using R (Data Science End-to-End Walkthrough)

Previous Lesson

Lesson 1: Prepare the Data (Data Science End-to-End Walkthrough)

Lesson 2-2 - Create Graphs and Plots Using R

4/19/2017 • 3 min to read • Edit Online

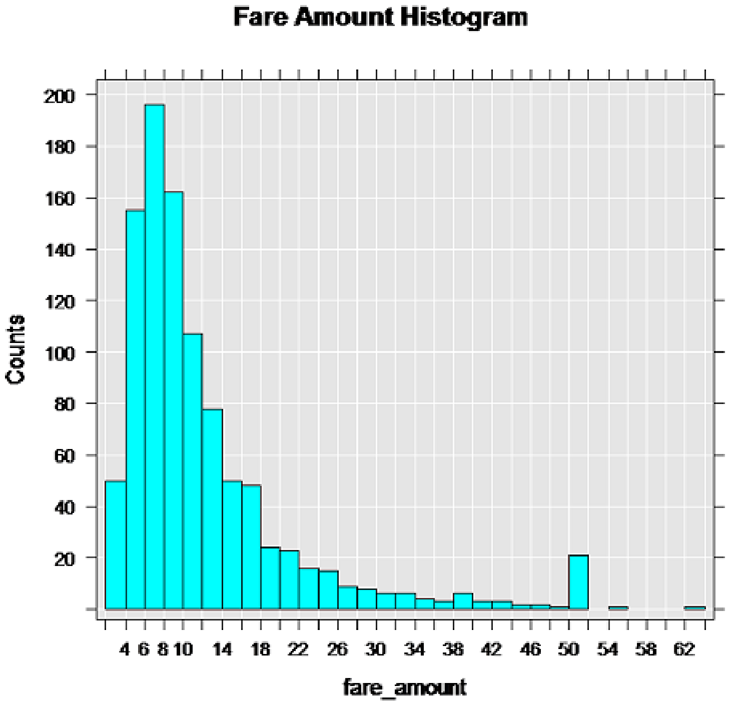
In this lesson, you'll learn techniques for generating plots and maps using R with SQL Server data. You'll create a simple histogram, to get some practice, and then develop a more complex map plot.

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 features.", 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.



> **NOTE**
>
> Because the ordering of rows using TOP is non-deterministic in the absence of an ORDER BY clause, you might see very different results. 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. This particular image was generated using about 10,000 rows of data.

Create a Map Plot

Typically, database servers block Internet access, so you might not be able to download the map representation and generate the plot you want. However, you can generate the map representation on the client, and then overlay on the map the points that are stored as attributes in the *nyctaxi_sample* table.

In other words, you'll create the map representation by calling into Google maps, and then pass the map representation to the SQL context.

This is a pattern that you might find useful when developing your own applications.

1. Define the function that creates the plot object.

   ```
   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 custom R function *mapPlot* creates a scatter plot that uses the taxi pickup locations to plot the number of rides that started from each location. It uses the **ggplot2** and **ggmap** packages, which should already be installed and loaded.
   - The *mapPlot* function takes two arguments: an existing data object, which you defined earlier using *RxSqlServerData*, and the map representation passed from the client.
   - Note the use of the *ds* variable to load data from the previously created data source, *inDataSource*. Whenever you use open source R functions, data must be loaded into data frames in memory. You can do this by using the *rxImport* function in the **RevoScaleR** package. However, this function runs in memory in the SQL Server context defined earlier. That is, the function is not using the memory of your local workstation.

2. Load the libraries required for creating the maps in your **local** R environment.

   ```
   library(ggmap)
   library(mapproj)
   gc <- geocode("Times Square", source = "google")
   googMap <- get_googlemap(center = as.numeric(gc), zoom = 12, maptype = 'roadmap', color = 'color';
   ```

   - This code is run on the R client. Note the repeated call to the libraries **ggmap** and **mapproj**. The previous function definition ran in the server context and the libraries were never loaded locally. Now you are bringing the plotting operation back to your workstation.

   - 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. Execute the plotting function and render the results in your local R environment, by wrapping the plot function in *rxExec* as shown here.

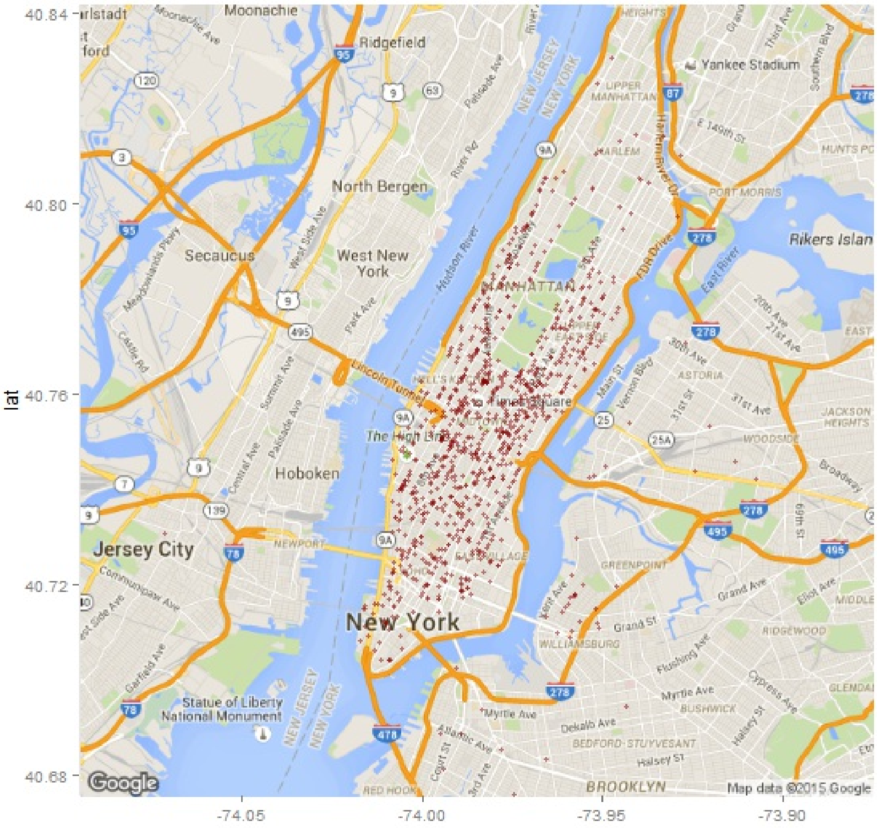
   ```
   myplots <- rxExec(mapPlot, inDataSource, googMap, timesToRun = 1)
   plot(myplots[[1]][["myplot"]]);
   ```

   - The *rxExec* function is part of the **RevoScaleR** package, and supports execution of arbitrary R functions in a remote compute context.
   - In the first line, the map data is passed as an argument (*googMap*) to the remotely executed function *mapPlot*. That is because the maps were generated in your local environment, and must be passed to the function in order to create the plot in the context of SQL Server.

   - The rendered data is then serialized back to the local R environment so that you can view it, using the

Plot window in RStudio or other R graphics device.

4. The following image shows the output plot. The taxi pickup locations are added to the map as red dots.



Next Lesson

Lesson 3: Create Data Features (Data Science End-to-End Walkthrough)

Lesson 3: Create Data Features (Data Science End-to-End Walkthrough)

4/19/2017 • 6 min to read • Edit Online

Data engineering is an important part of machine learning. Data often needs to be transformed 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'll compute the direct linear distance between two points, by using the haversine formula.

You'll compare two different methods for creating a feature from data:

- Using R and the *rxDataStep* function
- Using a custom function in Transact-SQL

For both methods, the result of the code is a SQL Server data source object, *featureDataSource*, that includes 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.

- You'll create a new R function, *ComputeDist*, to calculate the linear distance between two points specified by latitude and longitude values.
- You'll call the function to transform the data in the SQL Server data object you created earlier, an save it in a new data source, *featureDataSource*.

1. Run the following code to create a custom R function, *ComputeDist*. It takes in two pairs of latitude and longitude values, and calculates the linear distance between them. The function returns a 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.

Having defined the function, you will apply it to the data to create a new feature column, *direct_distance*.

2. Create a data source to work with by using the *RxSqlServerData* constructor.

```
featureDataSource = RxSqlServerData(table = "features",
   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)
```

3. Call the *rxDataStep* function to apply the `env$ComputeDist` function to the specified data.

```
start.time <- proc.time()

rxDataStep(inData = inDataSource, outFile = featureDataSource,
     overwrite = TRUE,
     varsToKeep=c("tipped", "fare_amount", passenger_count", "trip_time_in_secs",
        "trip_distance", "pickup_datetime", "dropoff_datetime", "pickup_longitude",
        "pickup_latitude", "dropoff_longitude", "dropoff_latitude")
     , transforms = list(direct_distance=ComputeDist(pickup_longitude,
        pickup_latitude, dropoff_longitude, dropoff_latitude)),
        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 can modify data in place. The arguments include a character vector of columns to pass through (*varsToKeep*), and a list that defines transformations.
- Any columns that are transformed are automatically output and therefore do not need to be included in the *varsToKeep* argument.
- Alternatively, you can specify that all columns in the source be included except for the specified variables, by using the *varsToDrop* argument.

4. Call *rxGetVarInfo* to inspect the schema of the new data source:

```
rxGetVarInfo(data = featureDataSource)
```

Results

"It takes CPU Time=0.74 seconds, Elapsed Time=35.75 seconds to generate features."
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: pickup_longitude, Type: numeric
Var 9: pickup_latitude, Type: numeric
Var 10: dropoff_longitude, Type: numeric
Var 11: dropoff_latitude, Type: numeric
Var 12: direct_distance, Type: numeric

Featurization using Transact-SQL

Now you'll create a custom SQL function, *ComputeDist*, to do the same thing as the R function you just created. The custom SQL function *ComputeDist* operates on an existing *RxSqlServerData* data object to create the new distance features from the existing latitude and longitude values.

You'll save the results of the transformation to a SQL Server data object, *featureDataSource*, just as you did using R.

1. Define a new custom SQL function, named *fnCalculateDistance*.

```
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
-- User-defined function 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
```

- 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.

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. To use the custom SQL function in R code, save the feature engineering 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_joined_1_percent
    tablesample (1 percent) repeatable (98052)"
```

> **TIP**
>
> This query is slightly different from the Transact-SQL query used earlier. It has been modified to get a smaller sample of data, to make this walkthrough faster.

4. 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)
```

5. Now that the new feature is created, call *rxGetVarsInfo* to create a summary of the feature table.

```
rxGetVarInfo(data = featureDataSource)
```

Comparing R Functions and SQL Functions

As it turns out, for this particular task, the Transact-SQL function approach is faster than the custom R function. Therefore, you'll use the Transact-SQL function for these calculations in subsequent steps.

Proceed to the next lesson to learn how to build a predictive model using this data and save the model to a SQL Server table.

> **TIP**
>
> Very often, feature engineering using Transact-SQL will be faster than R. For example, T-SQL includes windowing and ranking functions that are extremely fast in tasks that data scientists frequently perform in R, such as rolling moving averages and *n*tiles. Choose the most efficient method based on your data and task.

Next Lesson

Lesson 4: Build and Save the Model (Data Science End-to-End Walkthrough)

Previous Lesson

Lesson 2: View and Explore the Data (Data Science End-to-End Walkthrough)

Lesson 4: Build and Save the Model (Data Science End-to-End Walkthrough)

4/19/2017 • 6 min to read • Edit Online

In this lesson, you'll learn how to build a machine learning model and save the model in SQL Server.

Creating a Classification Model

The model you'll 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, `featureDataSource,` to train the tip classifier, using logistic regression.

Here are the features you'll use in the model:

- passenger_count
- trip_distance
- trip_time_in_secs
- direct_distance

Create the model using rxLogit

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 = featureDataSource))
   ```

   - 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'll want to 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, define the data object to use for storing the scoring results

```
scoredOutput <- RxSqlServerData(
  connectionString = connStr,
  table = "taxiScoreOutput"  )
```

- To make this example simpler, the input to the logistic regression model is the same `featureDataSource` 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 are saved in the table, *taxiscoreOutput*. Notice that the schema for this table is not defined when you create it using `rxSqlServerData`, but is obtained from the *scoredOutput* object output from `rxPredict`.

- To create the table that stores the predicted values, the SQL login running the `rxSqlServer` data function must have DDL privileges in the database. If the login cannot create tables, the statement will fail.

2. Call the *rxPredict* function to generate results.

```
rxPredict(modelObject = logitObj,
    data = featureDataSource,
    outData = scoredOutput,
    predVarNames = "Score",
    type = "response",
    writeModelVars = TRUE, overwrite = TRUE)
```

Plotting Model Accuracy

To get an idea of the accuracy of the model, you can use the *rxRocCurve* function to plot the Receiver Operating Curve. Because *rxRocCurve* is one of the new functions provided by the RevoScaleR package that supports remote compute contexts, you have two options:

- You can use the `rxRocCurve` function to execute the plot in the remote computer 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 use both techniques.

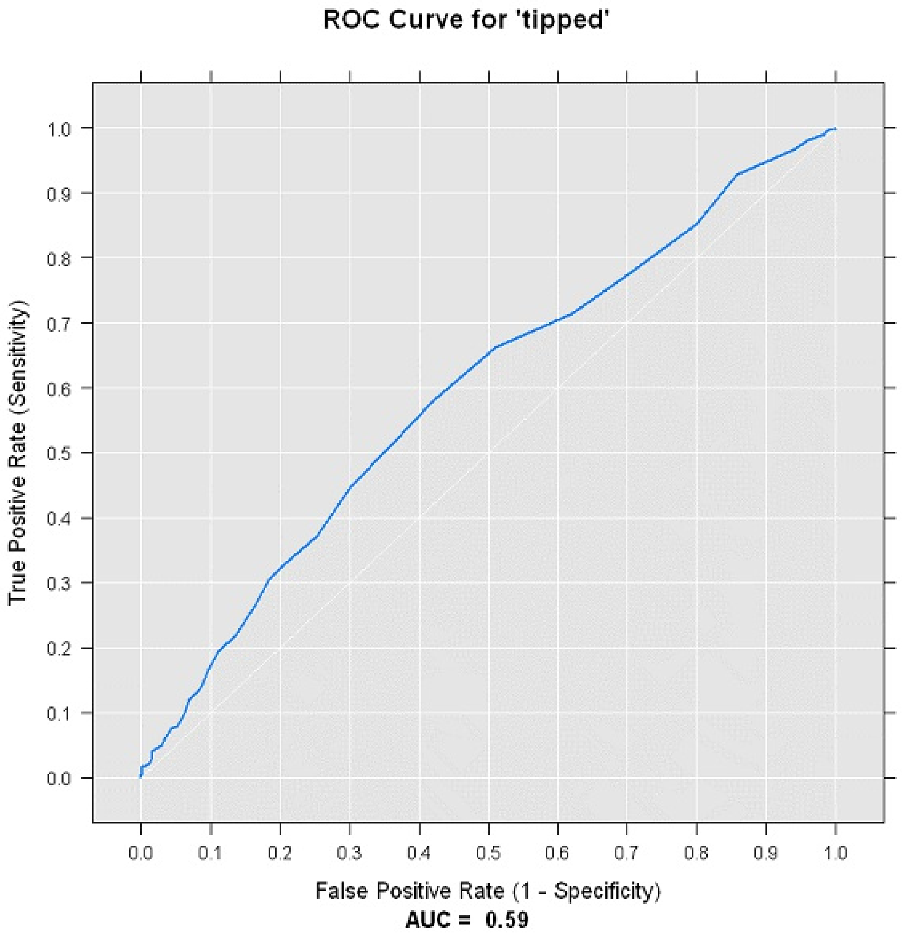
Execute a plot in the remote (SQL Server) compute context

1. Call the function *rxRocCurve* and provide the data defined earlier as input.

```
rxRocCurve( "tipped", "Score", scoredOutput)
```

Note that you must also specify the label column tipped (the variable you are trying to predict) and the name of the column that stores the prediction (*Score*).

2. View the graph that is generated by opening the R graphics device, or by clicking the **Plot** window in RStudio.



The graph is created on the remote compute context, and then returned to your R environment.

Create the plots in the local compute context using data from SQL Server

1. Use the *rxImport* function to bring the specified data to your local R environment.

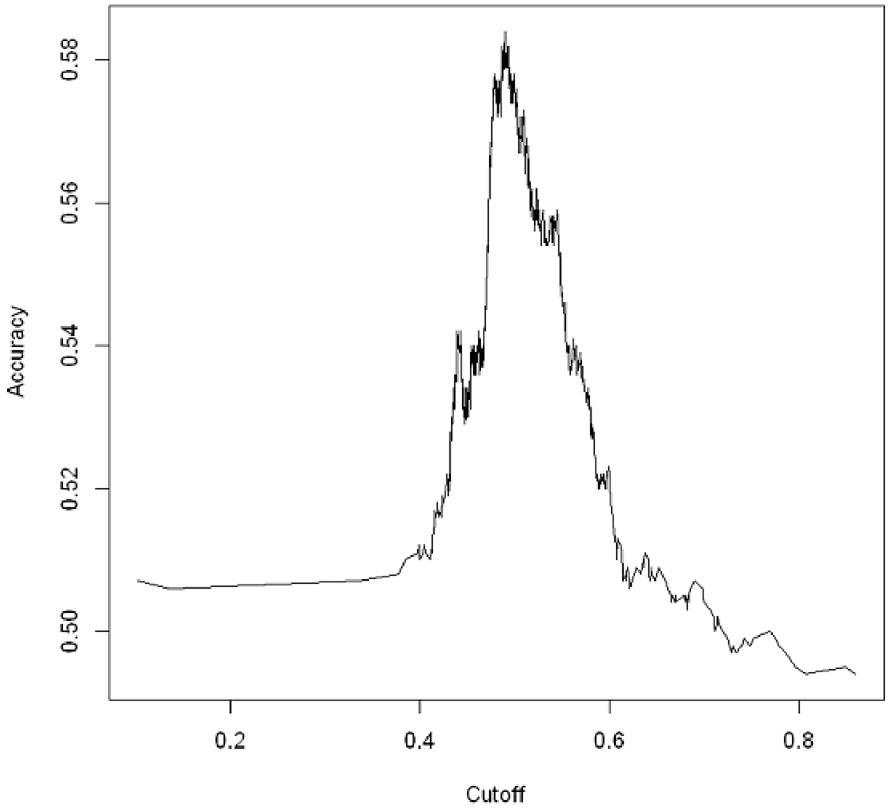
```
scoredOutput = rxImport(scoredOutput)
```

2. Having loaded the data into local memory, you can then call the **ROCR** library to create some predictions, and generate the plot.

```
library('ROCR')
pred <- prediction(scoredOutput$Score, scoredOutput$tipped)

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]
```

3. The following plot is generated in both cases.



Deploying a Model

After you have built a model and ascertained that it is performing well, you might want to *operationalize* the model. Because R Services (In-Database) lets you invoke an R model using a Transact-SQL stored procedure, it is extremely 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 will learn how to persist the model, and how to call it to make predictions.

Serialize the model

- In your local R environment, serialize the model and save it in a variable.

  ```
  modelbin <- serialize(logitObj, NULL)
  modelbinstr=paste(modelbin, collapse="")
  ```

  The *serialize* function is included in the R **base** package, and provides a simple low-level interface for serializing to connections. For more information, see http://www.inside-r.org/r-doc/base/serialize.

Move the model to SQL Server

- Open an ODBC connection, and call a stored procedure to store the binary representation of the model in a column in the database.

```
library(RODBC)
conn <- odbcDriverConnect(connStr )

# persist model by calling a stored procedure from SQL
q\<-paste("EXEC PersistModel @m='", modelbinstr,"'", sep="")
sqlQuery (conn, q)
```

Saving a model to a table requires only an INSERT statement. However, to make it easier, here we have used the *PersistModel* stored procedure is used.

For reference, here is the complete code of the stored procedure:

```
CREATE PROCEDURE [dbo].[PersistModel]  @m nvarchar(max)
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;
        insert into nyc_taxi_models (model) values (convert(varbinary(max),@m,2))
END
```

TIP

We recommend creating helper functions such as this stored procedure to make it easier to manage and update your R models in SQL Server.

Invoke the saved model

After you have saved the model to the database, you can call it directly from Transact-SQL code, using the system stored procedure, sp_execute_external_script (Transact-SQL).

For example, to generate predictions, you simply connect to the database and run a stored procedure that uses the saved model as an input, together with some input data.

However, if you have a model you use often, it's easier to wrap the input query and the call to the model, as well as any other parameters, in a custom stored procedure.

In the next lesson, you'll learn how to perform scoring against the saved model using Transact-SQL.

Next Lesson

Lesson 5: Deploy and Use the Model (Data Science End-to-End Walkthrough)

Previous Lesson

Lesson 3: Create Data Features (Data Science End-to-End Walkthrough)

Lesson 5: Deploy and Use the Model (Data Science End-to-End Walkthrough)

4/19/2017 • 5 min to read • Edit Online

In this lesson, you will use your R models in a production environment, by wrapping the persisted model in 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.

There are two different ways that you can call a model for scoring:

- **Batch scoring mode** lets you create multiple predictions based on input from a SELECT query.
- **Individual scoring mode** lets you create predictions one at a time, by passing a set of feature values for an individual case to the stored procedure, which returns a single prediction or other value as the result.

You'll learn how to create predictions using both the individual scoring and batch scoring methods.

Batch Scoring

For convenience, you can use a stored procedure that was created when you initially ran the PowerShell script in Lesson 1. This stored procedure 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 will get a tip

Use the stored procedure PredictTipBatchMode

1. Take a minute to look over the script that defines 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
```

- Note the SELECT statement that calls the stored model. You can store any trained model in a SQL table, by using a column of type **varbinary(max)**. In this code, the model is retrieved from the table, stored in the SQL variable *@lmodel2*, and passed as the parameter *mod* to the system stored procedure sp_execute_external_script (Transact-SQL).

- The input data used for scoring is passed as a string to the stored procedure.

  To define input data for this particular model, create a query that returns valid data. As data is retrieved from the database, it is stored in a data frame called *InputDataSet*. All the rows in this data frame are used for batch scoring.

  - *InputDataSet* is the default name for input data to the sp_execute_external_script (Transact-SQL) procedure; you can define another variable name if needed.
  - To generate the scores, the stored procedure calls the *rxPredict* function from the **RevoScaleR** library.
- The return value for the stored procedure, *Score*, is a predicted probability that the driver will be given a tip.

2. Optionally, you could easily apply some kind of filter to the returned values to categorize the return values into "yes - tip " or "no tip" groups. For example, a probability of less than 0.5 would mean no tip is likely.

3. Call the stored procedure in batch mode:

```
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="")
sqlQuery (conn, q)
```

Single Row Scoring

Instead of using a query to pass the input values to the saved R model, you might want to provide the features as arguments to the stored procedure.

Use the stored procedure PredictTipSingleMode

1. Take a minute to review the following code is for the stored procedure, *PredictTipSingleMode*, which should already be created in your database.

```
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 feature values as input, such as passenger count and trip distance, scores these features using the stored R model, and outputs a score.

Call the stored procedure and pass parameters

1. In SQL Server Management Studio, you can use the Transact-SQL **EXEC** to call the stored procedure, and pass it the required inputs. .

```
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*.

2. To run this same call from R code, you simply define an R variable that contains the entire stored procedure call.

```
q = "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*.

Generate scores

1. Call the *sqlQuery* function of the **RODBC** package, and pass the connection string and the string variable containing the the stored procedure call.

```
# predict with stored procedure in single mode
sqlQuery (conn, q)
```

For more information about **RODBC**, see http://www.inside-r.org/packages/cran/RODBC/docs/sqlQuery.

Conclusion

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 some additional models based on this data set. For example, you might try creating models like these:

- A regression model that predicts the tip amount
- A multiclass classification model that predicts whether the tip will be 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
  Lesson 4: Build and Save the Model (Data Science End-to-End Walkthrough)

See Also

SQL Server R Services Tutorials

Data Science Deep Dive: Using the RevoScaleR Packages

5/22/2017 • 4 min to read • Edit Online

This tutorial demonstrates how to use the enhanced R packages provided in R Services (In-Database) to work with SQL Server data and create scalable R solutions, by using the server as a compute context for high-performance big data analytics.

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 will also learn how to analyze and plot data both locally and on the remote server, and how to create and deploy models.

> **NOTE**
>
> This tutorial works specifically with SQL Server data on Windows, and uses in-database compute contexts. If you want to use R in other contexts, such as Teradata, Linux, or Hadoop, see these Microsoft R Server tutorials:
>
> - Use R Server with sparklyr
> - Explore R and ScaleR in 25 functions
> - Get Started with ScaleR on Hadoop MapReduce RevoScaleR Teradata Getting Started Guide

Overview

To illustrate the flexibility and processing power of the ScaleR packages, in this tutorial you'll move data and swap compute contexts frequently.

- Data is initially obtained from CSV files or XDF files. You'll import the data into SQL Server using the functions in the RevoScaleR package.
- Model training and scoring will be performed in the SQL Server compute context. You'll create new SQL Server tables, using the **rx** functions, to save your scoring results.
- You'll create plots both on the server and in the local compute context.
- To train the model, you will use data already stored in a SQL Server database. All computations will be performed on the SQL Server instance.
- You'll extract a subset of data and save it as an XDF file for re-use in analysis on your local workstation.
- New data used during the scoring process is extracted from the SQL Server database using an ODBC connection. All computations are performed on the local workstation.
- Finally, you'll perform a simulation based on a custom R function, using the server compute context.

Get Started Now

This tutorial takes about 75 minutes to complete, not including setup.

1. Work with SQL Server Data using R
2. Create SQL Server Data Objects using RxSqlServerData
3. Query and Modify the SQL Server Data
4. Define and Use Compute Contexts
5. Create and Run R Scripts
6. Visualize SQL Server Data using R
7. Create R Models

8. Score New Data
9. Transform Data Using R
10. Load Data into Memory using rxImport
11. Create New SQL Server Table using rxDataStep
12. Perform Chunking Analysis using rxDataStep
13. Analyze Data in Local Compute Context;
14. Move Data between SQL Server and XDF File
15. Create a Simple Simulation

Target Audience

This tutorial is intended for data scientists or for people who are already somewhat familiar with R and data science tasks including exploration, statistical analysis, and model tuning. However, all the code is provided, so even if you are new to R, you can easily 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 SQL Server Management Studio or other database tools, such as Visual Studio.

> **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 2016 and enable R Services (in-Database). Or, install SQL Server 2017 and enable Machine Learning Services and choose the R language. The setup process is described in SQL Server 2016 Books Online.

- **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**

  You must also install the RevoScaleR packages and related providers in your R development environment. The easiest way to do this is to install Microsoft R Client or Microsoft R Server (Standalone). For more information, see Set Up a Data Science Client

  > **NOTE**
  >
  > Other versions of Revolution R Enterprise or Revolution R Open are not supported.
  >
  > An open source distribution of R, such as R 3.2.2, will not work in this tutorial, because only the RevoScaleR functions can use remote compute contexts.

- **Additional R Packages**

  For this tutorial, you will need to 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 computer you use for R solution development, and on the SQL Server computer where R scripts will be run. 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.** It is important that the packages be installed in the R package library that is used by the SQL Server instance.

For more information, see Prerequisites for Data Science Walkthroughs.

Next Step

Work with SQL Server Data using R

Work with SQL Server Data using R

5/22/2017 • 4 min to read • Edit Online

In this lesson, you'll 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'll 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.
- Learn how to enable tracing on the remote compute context.

Create the Database and User

For this walkthrough, you'll create a new database in SQL Server 2017, and add a SQL login with permissions to write and read data, as well as to run R scripts.

> **NOTE**
>
> If you are only reading data, the account that runs the R scripts requires only SELECT permissions (**db_datareader** role) on the specified database. However, in this tutorial, you will need 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 will need the permission, EXECUTE ANY EXTERNAL SCRIPT,to be able 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 are provided in this article.

- **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 will be prefixed with this *schema*. 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 username, the user is called 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 will be created with the 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 generally called out in the tutorial.

  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

5/22/2017 • 6 min to read • Edit Online

Now that you have created the SQL Server database and have the necessary permissions to work with the data, you'll create objects in R that let you work with the data — both on the server and on your workstation.

Create the SQL Server Data Objects

In this step, you'll use R to create and populate two tables. Both tables contain simulated credit card fraud data. One table is used for training the models, and the other table is used for scoring.

To create tables on the remote SQL Server computer, you'll use the **RxSqlServerData** function provided in the **RevoScaleR** package.

> **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. Provide your database connection string in an R variable. Here we've provided two examples of valid ODBC connection strings for SQL Server: one using a SQL login, and one for Windows integrated authentication (recommended).

   Using a SQL login

   ```
   sqlConnString <- "Driver=SQL Server;Server=instance_name; Database=DeepDive;Uid=user_name;Pwd=password"
   ```

   Using 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 accumulate 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.

For this walkthrough, you'll use the batch process size defined by the SQL Server instance to control the number of rows in each chunk, and save that value in the variable *sqlRowsPerRead*. We recommend that you experiment with this setting on your system when you are working with a large data set.

4. 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

You'll create the table that holds the scoring data using the same process.

1. Create a new R variable, *sqlScoreTable*, to store the name of the table used for scoring.

```
sqlScoreTable <- "ccFraudScoreSmall"
```

2. 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; just specify a different table name.

Later in this tutorial you'll 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, you'll 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 utility function, **rxGetOption**. This function is provided in the **RevoScaleR** package to help you set and manage options related to local and remote compute contexts, such as the default shared directory, the number of processors (cores) to use in computations, etc. This call is useful because it 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'll 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 yet. 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. You'll follow the same 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 will be 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

The **rxDataStep** is a powerful function that can perform multiple transformations on an R data frame, to convert the data into the representation required by the destination. In this case, the destination is SQL Server.

You can also specify transformations on the data, by using R functions in the arguments to rxDataStep. You'll see examples of these operations later.

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

5/22/2017 • 3 min to read • Edit Online

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'll 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.

```
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 will provide a string vector containing the abbreviations, and then map these categorical values to the original integer identifiers. After this varaible is ready, you'll use it in the *colInfo* argument, to specify that this column be handled as a factor. Thereafter, the abbreviations will be used and the column handled as a factor whenever this data is analyzed or imported.

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.
   - Mapping the column to abbreviations before using it as a factor actually improves performance as well. For more information, see R and Data Optimization

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

5/26/2017 • 3 min to read • Edit Online

Suppose that you want to perform some of the more complex calculations on the server, rather than your local computer. To do this, you can create a compute context, which lets R code run on the server.

The **RxInSqlServer** function is one of the enhanced R functions provided in the RevoScaleR package. The function handles the tasks of creating the database connection and passing objects between the local computer and the remote execution context.

In this step, you'll learn how to use the **RxInSqlServer** function to define a compute context in your R code.

To create a compute context requires the following basic information about the SQL Server instance :

- The connection string for the instance
- A specification for how output should be handled
- Optional arguments for enabling tracing, or for specifying a shared directory

Create and Set a Compute Context

1. Specify the connection string for the instance where computations will take place. This is just one of several variables that you'll pass to the *RxInSqlServer* function to create the compute context. You can re-use the connection string that you created earlier, or create a different one if you want to move the computations to a different server or use a different identity.

   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 will be blocking and will not return until it has completed or has failed. For more information, see Distributed and parallel computing in Microsoft R.

   - **FALSE**. Jobs will be 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. 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.

   ```
   dir.create(sqlShareDir, recursive = TRUE)
   ```

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)
   ```

   You might observe that the syntax for RxInSqlServer* is 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, we'll use the compute context that does not have tracing enabled. That is because performance for the tracing-enabled option has not been tested for all operations.

However, if you decide to use tracing, be aware that your experience may be affected by network connectivity.

Now that you have created a remote compute context, you'll learn how to change 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

5/22/2017 • 4 min to read • Edit Online

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'll 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 will 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 Summary Statistics

To see how the compute context works, try generating some summary statistics using the *sqlFraudDS* data source. Remember, the data source object just defines the data you'll 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 more information about similar functions, see Data Summaries in the ScaleR reference.

2. 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, so you decide to add the high and low values to the RxSqlServerData data source.

Fortunately R Services (In-Database) includes optimized functions that can very 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.

   You'll also add to 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, you can 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 in *ccColInfo*.

   These 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'll 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

5/22/2017 • 2 min to read • Edit Online

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 will 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.

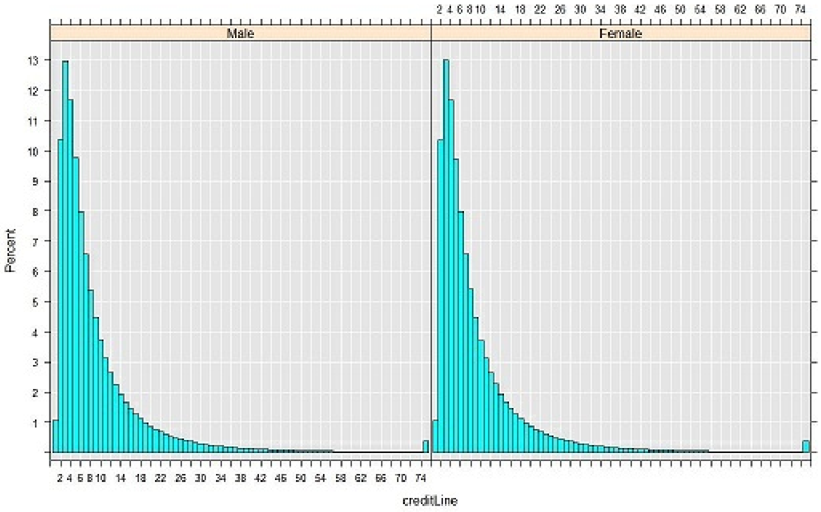
```
rxHistogram(~creditLine|gender, data = sqlFraudDS,  histType = "Percent")
```

Internally, rxHistogram calls the rxCube function, which is included in the **RevoScaleR** package. The rxCube function 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 will automatically be used in the histogram.

2. The return value of rxCube is by default 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.

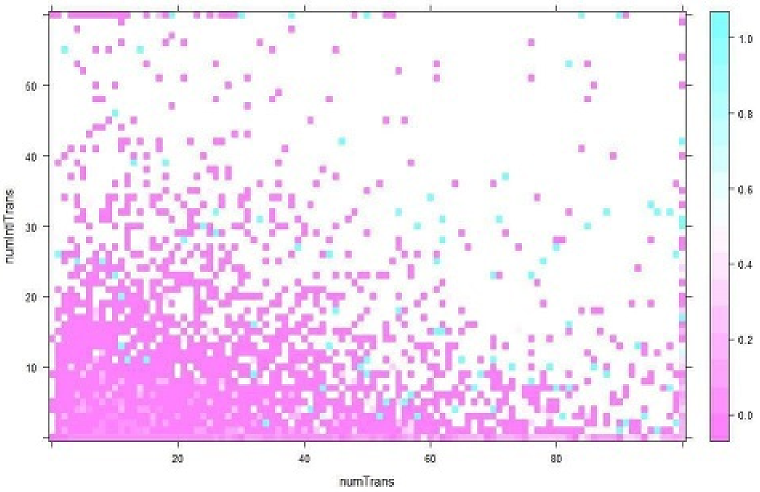
```
cubePlot <- rxResultsDF(cube1)
```

> **TIP**
>
> Note that 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.

Next Step

Create Models

Previous Step

Lesson 2: Create and Run R Scripts

Create R Models

5/22/2017 • 3 min to read • Edit Online

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

You'll 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, you'll use the **rxLinMod** function, which supports remote compute contexts.

1. Create an R variable to store the completed model, and call the *rxLinMod* function, passing an appropriate formula.

   ```
   linModObj <- rxLinMod(balance ~ gender + creditLine,  data = sqlFraudDS)
   ```

2. To view a summary of the results, you can 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

Now, you'll create a logistic regression model that indicates whether a particular customer is a fraud risk. You'll use the *rxLogit* function, included in the **RevoScaleR** package, 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

5/22/2017 • 3 min to read • Edit Online

Now that you have a model you can use for predictions, you'll feed it data from the SQL Server database to generate some predictions.

You will use the logistic regression model, *logitObj*, to create scores for another data set that uses the same independent variables as inputs.

> **NOTE**
>
> You will 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, you'll create a new data source object, and use that 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 function executes the DDL and returns TRUE if the table is successfully dropped, FALSE otherwise.

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 will be on 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, you can 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 will compute and display a histogram of the 10,000 predicted scores. Computation will be faster if you specify the low and high values, so you'll 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 minVal,",
    "MAX(ccFraudLogitScore) AS maxVal 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

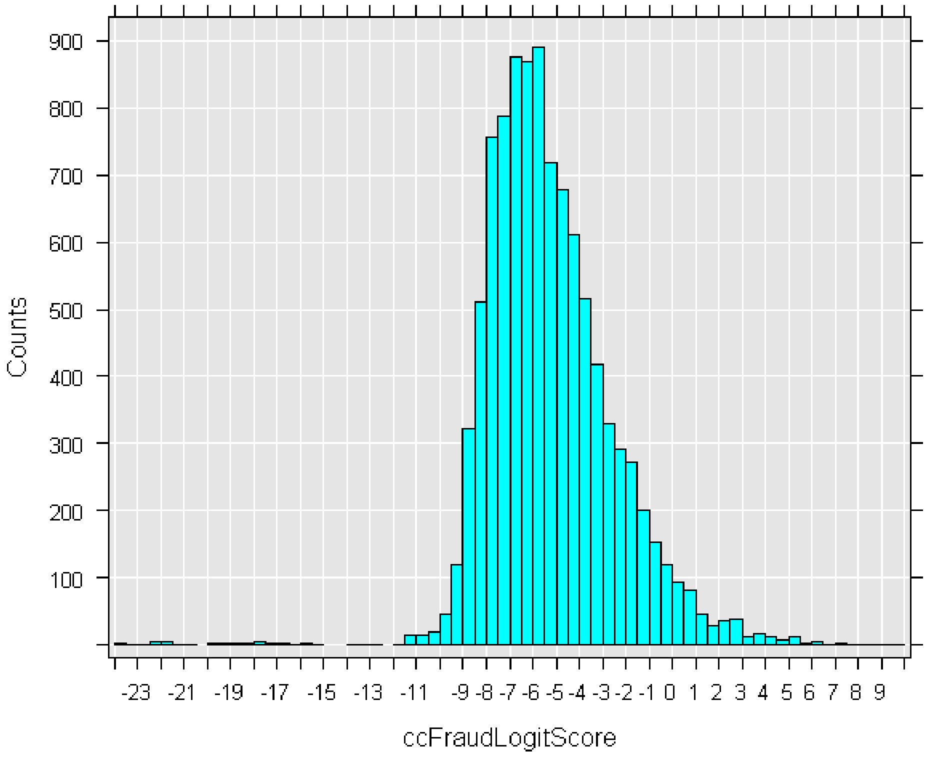
3. Now that the maximum and minimum values are available, use the values to create the score data source.

```
sqlOutScoreDS <- RxSqlServerData(sqlQuery = "SELECT ccFraudLogitScore FROM ccScoreOutput",
    connectionString = sqlConnString,
    rowsPerRead = sqlRowsPerRead,
        colInfo = list(ccFraudLogitScore = list(
            low = floor(minMaxVals[1]),
                    high = ceiling(minMaxVals[2]) ) ) )
```

4. Finally, use the score data source object to get the scoring data, 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

5/22/2017 • 2 min to read • Edit Online

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'll 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, you'll use a function from another R package to transform your 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, you'll 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, you will get 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 Transforming and Subsetting Data

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

5/22/2017 • 2 min to read • Edit Online

The **rxImport** function can be used to move data from a data source into a data frame in R 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'll 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 will get the information about just the high-risk customers, and load it 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. You use the function **rxImport** to actually load 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: Rows Read: 35, Total Rows Processed: 35, Total Chunk Time: 0.036 seconds

4. Now that you have the high-risk observations in a data frame in memory, you can use various R functions to manipulate the data frame. For example, you can order customers by their risk score, and print 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 ccFraudProb1

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, and you can 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

See Also

Machine Learning Tutorials

Create New SQL Server Table using rxDataStep

5/22/2017 • 3 min to read • Edit Online

In this lesson, you'll learn how to move data between in-memory data frames, the SQL Server context, and local files.

NOTE

For this lesson, you'll use a different data set. The Airline Delays dataset is a public dataset that is widely used for machine learning experiments. If you're just getting started with R, this dataset is handy to keep around for testing, as it is used in various product samples for R Services (In-Database) that were published with SQL Server 2017. The data files you'll need for this example are available in the same directory as other product samples.

Create SQL Server Table from Local Data

In the first part 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.

In this lesson, you'll use a different approach, and get data from a file saved in the XDF format. The XDF format is an XML standard developed for high-dimensional data. 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.

After doing some lightweight transformations on the data using the XDF file, you'll save the transformed data into a new SQL Server table.

NOTE

You'll need DDL permissions for this step.

1. Set the compute context to the local workstation.

```
rxSetComputeContext("local")
```

2. Define a new data source object using the **RxXdfData** function. For an XDF data source, you just specify the path to the data file. You could specify the path to the file using a text variable, but in this case, there's a handy shortcut, because the sample data file (AirlineDemoSmall.xdf) is in the directory returned by the rxGetOption function.

```
xdfAirDemo <- RxXdfData(file.path(rxGetOption("sampleDataDir"),  "AirlineDemoSmall.xdf"))
```

3. 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. For more information about XDF files, see the ScaleR Getting Started Guide

1. Now, you'll 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 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 will you see the final status message: *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* argumrnt 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

5/4/2017 • 2 min to read • Edit Online

The **rxDataStep** function can be used 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 way it works is that you read the data in chunks and use R functions to process each chunk of data in turn, and then write the summary results for each chunk to a common SQL Server data source.

In this lesson, you'll practice this technique by using the `table` function in R, to compute a contingency table.

> **TIP**
>
> 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. First, create a custom R function that calls the *table* function on each chunk of data, and name it `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. You'll define a SQL Server data source to hold the data you're processing. Start by assigning a SQL query to a variable.

```
dayQuery <-  "SELECT DayOfWeek FROM AirDemoSmallTest"
```

4. Plug that variable into the *sqlQuery* argument of a new SQL Server data source.

```
inDataSource <- RxSqlServerData(sqlQuery = dayQuery,
    connectionString = sqlConnString,
    rowsPerRead = 50000,
    colInfo = list(DayOfWeek = list(type = "factor",
        levels = as.character(1:7))))
```

If you ran *rxGetVarInfo* on this data source, you'd see that it contains just the single column: *Var 1: DayOfWeek, Type: factor, no factor levels available*

5. 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, and 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) }
```

6. Now you'll 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)
```

7. To view 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 another 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 (Data Science Deep Dive)

5/22/2017 • 1 min to read • Edit Online

Although it might be faster to run complex R code using the server context, sometimes it is just more convenient to get your data out of SQL Server and analyze it on your private workstation.

In this section, you'll learn how to switch back to a local compute context, and move data between contexts to optimize performance.

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.

   ```
   sqlServerDS1 <- RxSqlServerData(
      connectionString = sqlConnString,
      table = sqlFraudTable,
      colInfo = ccColInfo,
      rowsPerRead = 10000)
   ```

   Previously, the value of *rowsPerRead* was set to 5000.

3. Now, 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

5/22/2017 • 3 min to read • Edit Online

When you are working in a local compute context, you have access to both local data files and the SQL Server database (defined as an RxSqlServerData data source).

In this section, you'll learn how to get data and store it in a file on the local computer so that you can perform transformations on the data. When you're done, you'll use the data in the file to create a new SQL Server table, by using rxDataStep.

Create a SQL Server Table from an XDF File

The rxImport function lets you import data from any supported data source to a local XDF file. Using a local file can be convenient if you want to do many different analyses on data that is stored in a SQL Server database, and you want to avoid running the same query over and over.

For this exercise, you'll 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 just these states on your local computer and work with the variables gender, cardholder, state, and balance.

1. Re-use the *stateAbb* vector you created earlier to identify the levels to include, and then print to the console the new variable, *statesToKeep*.

   ```
   statesToKeep <- sapply(c("CA", "OR", "WA"), grep, stateAbb)
   statesToKeep
   ```

   Results

CA	OR	WA
5	38	48

2. Now you'll define the data you want to bring over from SQL Server, using a Transact-SQL query. Later you'll 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, you'll 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 will return data for only three states. You can re-use 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))
        )
```

4. Set the compute context to **local**, because you want all the data available on your local computer.

   ```
   rxSetComputeContext("local")
   ```

5. Create the data source object by passing all the variables that you just defined as arguments to RxSqlServerData.

   ```
   sqlServerImportDS <- RxSqlServerData(
       connectionString = sqlConnString,
       sqlQuery = importQuery,
       colInfo = importColInfo)
   ```

6. Then, 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.

7. 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

8. 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:

   ```
   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'll create a simple simulation using a custom R function and run it on the remote server.

Next Step

Create a Simple Simulation

Previous Step

Analyze Data in Local Compute Context

Create a Simple Simulation

5/22/2017 • 3 min to read • Edit Online

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 node.

In this lesson, you'll 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 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 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 you'll use it only to run your custom function on the server.

1. Call the custom function as an argument to rxExec, along with some 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

TIP

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:

- Set the variables *ccFraudCsv* and *ccScoreCsv* to point to the new data files
- Change the name of the table referenced in *sqlFraudTable* to *ccFraud10*
- Change the name of the table referenced in *sqlScoreTable* to *ccFraudScore10*

Previous Step

Move Data between SQL Server and XDF File

In-Database R Analytics for SQL Developers

5/17/2017 • 3 min to read • Edit Online

The goal of this walkthrough is to provide SQL programmers with hands-on experience building a machine learning solution in SQL Server. In this walkthrough, you'll learn how to incorporate R into an application or BI solution by wrapping R code in stored procedures.

> **NOTE**
>
> The same solution is available in Python. SQL Server 2017 is required. See LINK.

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.

In this walkthrough, we'll assume that you have been given all the R code needed for the solution, and you'll focus on building and deploying the solution using SQL Server.

- Step 1: Download the Sample Data

  Download the sample dataset and the sample SQL 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 SQL Server 2017 instance and loads the sample data to the table.

- Step 3: Explore and Visualize the Data

  Perform basic data exploration and visualization, by calling R packages and functions from Transact-SQL stored procedures.

- Step 4: Create Data Features using T-SQL

  Create new data features using custom SQL functions.

- Step 5: Train and Save a Model using T-SQL

  Build and save the machine learning model, using stored procedures.

- Step 6: Operationalize the Model

  After the model has been saved to the database, call the model for prediction from Transact-SQL by using 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 RStudio or R Tools for Visual Studio. The R scripts provided in this tutorial have already been developed and debugged using traditional R tools.

If you are interested in learning how to develop R scripts that can run in SQL Server 2017, see this tutorial: Data Science End-to-End Walkthrough)

Scenario

This walkthrough uses the well-known NYC Taxi data set. To make this walkthrough easy and quick, 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.

Requirements

This walkthrough is intended for users who are already familiar with fundamental database operations, such as creating databases and tables, importing data into tables, and creating SQL queries. All R code is provided, so no R development environment is required. 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.

However, before starting the walkthrough, you must complete these preparations:

- Connect to an instance of SQL Server 2016 with R Services, or SQL Server 2017 with Machine Learning Services and R enabled.
- 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.

Next Step

Step 1: Download the Sample Data

See Also

SQL Server R Services Tutorials

SQL Server R Services

Set up SQL Server Machine Learning Services

Step 1: Download the Sample Data (In-Database Advanced Analytics Tutorial)

4/19/2017 • 1 min to read • Edit Online

In this step, you'll download the sample dataset and the Transact-SQL script files that are used in this walkthrough. 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 the Data and Scripts

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.

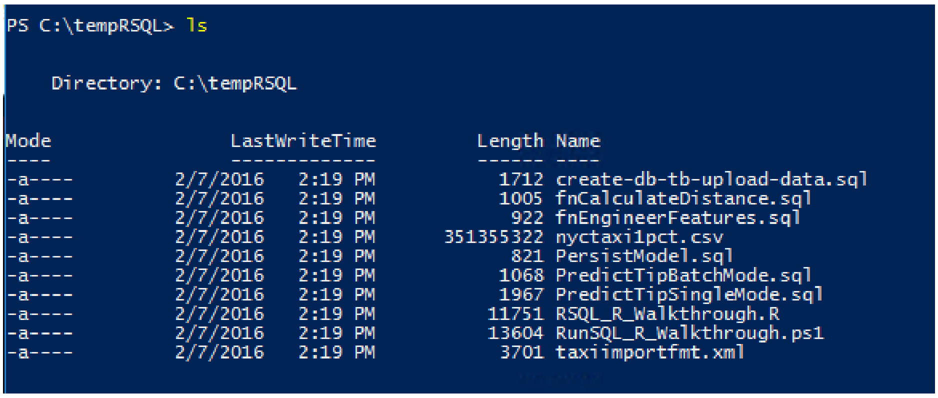
   > **TIP**
   >
   > If you get an error, you can temporarily set the policy for execution of PowerShell scripts to **unrestricted** only for this walkthrough, by using the Bypass argument and scoping the changes to the current session.
   >
   > ```
   > Set\-ExecutionPolicy Bypass \-Scope Process
   > ```
   >
   > 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.

   ```
   ls
   ```

   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 Step

Step 2: Import Data to SQL Server using PowerShell

Previous Step

In-Database Advanced Analytics for SQL Developers (Tutorial)

See Also

SQL Server R Services Tutorials

Step 2: Import Data to SQL Server using PowerShell

5/17/2017 • 3 min to read • Edit Online

In this step, you'll run one of the downloaded scripts, to create the database objects required for the walkthrough. The script also creates most of the stored procedures you'll use, and uploads the sample data to a table in the database you specified.

Run the Scripts to Create SQL Objects

Among the downloaded files you should see a PowerShell script. To prepare the environment for the walkthrough, you'll run this script.

Actions performed by the script include:

- Installing 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**.
- Creating a database and a table on the SQL Server instance, and bulk-inserting data into the table.
- Creating multiple SQL functions and stored procedures.

To run the script

1. Open a PowerShell command prompt as administrator and run the following command.

   ```
   .\RunSQL_SQL_Walkthrough.ps1
   ```

   You will be prompted to input the following information:

   - The name or address of a SQL Server 2017 instance where R Services (In-Database) has been installed
   - The user name and password for an account on the instance. The account must have permissions to create databases, create tables and stored procedures, and upload 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:\tempRSQL\nyctaxi1pct.csv`

2. As part of this step, all the Transact-SQL scripts are also modified to replace placeholders with the database name and user name that you provide as script inputs.

   Take a minute to review the stored procedures and functions created by the script.

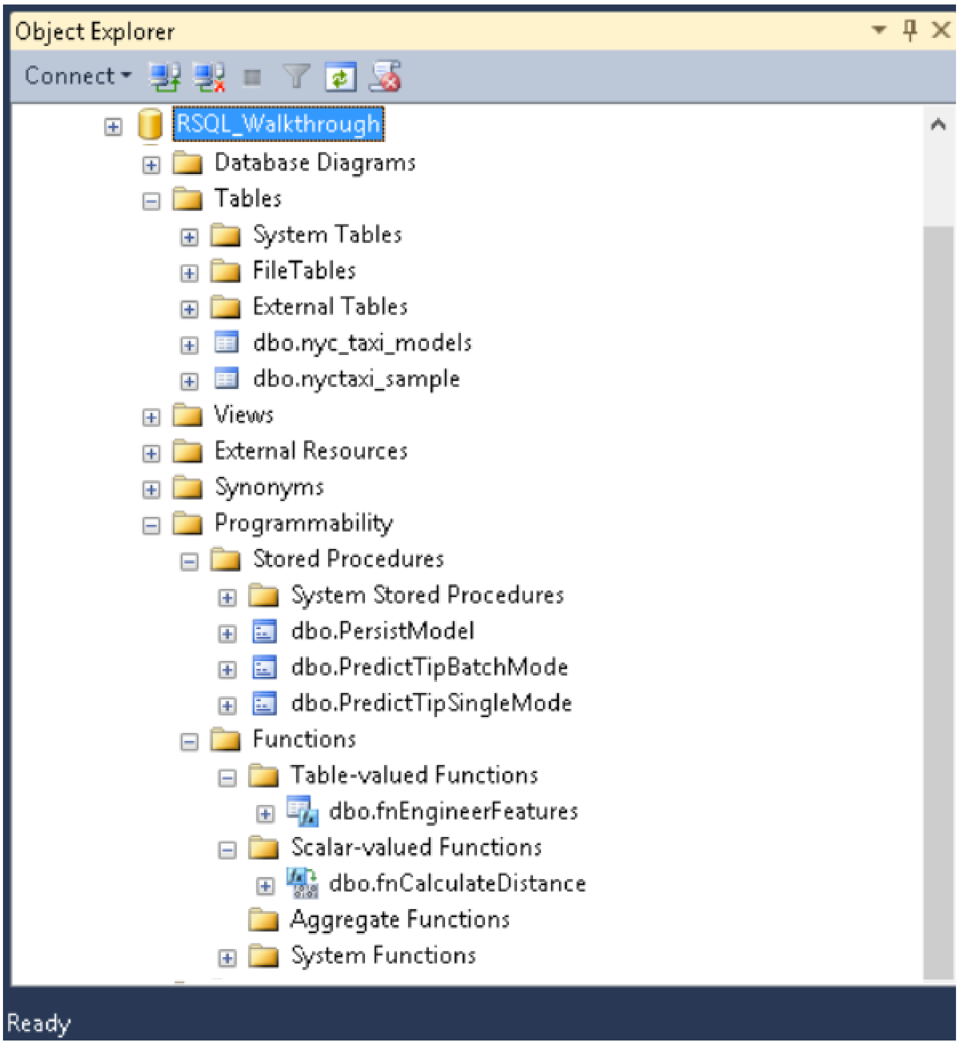
SQL SCRIPT FILE NAME	FUNCTION

SQL SCRIPT FILE NAME	FUNCTION
create-db-tb-upload-data.sql	Creates a database and two tables: 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. nyc_taxi_models: Used to persist the trained advanced analytics model.
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
PersistModel.sql	Creates a stored procedure that can be called to save a model. The stored procedure takes a model that has been serialized in a varbinary data type, and writes it to the specified table.
PredictTipBatchMode.sql	Creates a stored procedure that 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.
PredictTipSingleMode.sql	Creates a stored procedure that 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.

You'll create some additional stored procedures in the latter part of this walkthrough:

SQL SCRIPT FILE NAME	FUNCTION
PlotHistogram.sql	Creates a stored procedure for data exploration. This stored procedure calls an R function to plot the histogram of a variable and then returns the plot as a binary object.
PlotInOutputFiles.sql	Creates a stored procedure for data exploration. This stored procedure creates a graphic using an R function and then saves the output as a local PDF file.
TrainTipPredictionModel.sql	Creates a stored procedure that 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.

3. Log in to the SQL Server instance using SQL Server Management Studio and the login you specified, to verify that you can see the database, tables, functions, and stored procedures that were created.



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.

Next Step

Step 3: Explore and Visualize the Data

Previous Step

Step 1: Download the Sample Data

See Also

In-Database Advanced Analytics for SQL Developers (Tutorial)

SQL Server R Services Tutorials

Step 3: Explore and Visualize the Data (In-Database Advanced Analytics Tutorial)

5/17/2017 • 9 min to read • Edit Online

Developing a data science solution usually includes intensive data exploration and data visualization. In this step, you'll review the sample data, and then generate some plots using R functions. These R functions are already included in R Services (In-Database); in this walkthrough, you'll practice calling R functions from Transact-SQL.

Review the Data

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 Plots using R in T-SQL

Because visualization is such a powerful tool for understanding the distribution of the data and outliers, R provides many packages for visualizing data. The standard open-source distribution of R also includes many functions for crating histograms, scatter plots, box plots, and other data exploration graphs.

R typically creates images using an R device for graphical output. You can capture the output of this device and store the image in a **varbinary** data type for rendering in application, or you can save the images to any of the support file formats (.JPG, .PDF, etc.).

In this section, you'll learn how to work with each type of output using stored procedures.

- Storing plots as varbinary data type
- Saving plots in files (.JPG, .PDF) on the server

Storing plots as varbinary data type

You'll use `rxHistogram`, one of the enhanced R functions provided in R Services (In-Database), to plot a histogram based on data from a Transact-SQL query. To make it easier to call the R function, you will wrap it in a stored procedure, *PlotHistogram*.

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.

To create the stored procedure PlotHistogram

1. In SQL Server Management Studio, open a new query window.
2. Select the database for the walkthrough, and create the procedure using this statement.

```
CREATE PROCEDURE [dbo].[PlotHistogram]
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
```

Be sure to modify the code to use the correct table name, if needed.

- 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**.

  In R, when you issue a high-level plotting command, R will open 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. This is a temporary workaround for CTP3.

To output varbinary data to viewable graphics file

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:

   ```
   bcp "exec PlotHistogram" queryout "plot.jpg" -S <SQL Server instance name> -d  <database name>  -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]:
   ```

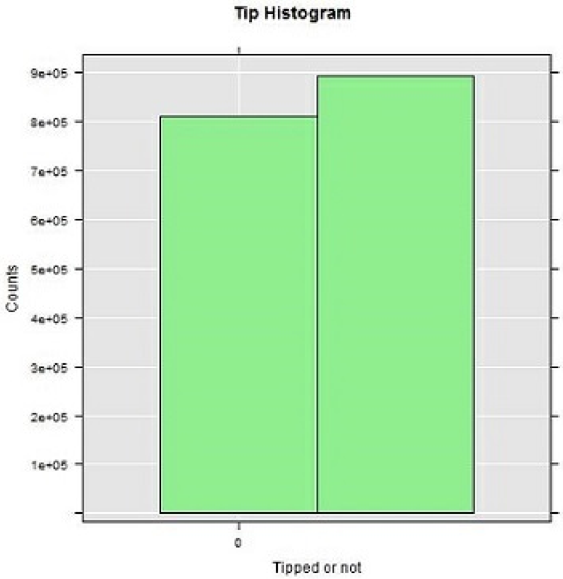
   Results

   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.



Saving plots in files (jpg, pdf) on the server

Outputting an R plot to a binary data type might be convenient for consumption by applications, but it is not very useful to a data scientist who needs the rendered plot during the data exploration stage. Typically the data scientist will generate multiple data visualizations, to get insights into the data from different perspectives.

To generate graphs that can be more easily viewed, you can use a stored procedure that creates the output of R in popular formats such as .JPG, .PDF, and .PNG. After the stored procedure creates the graphic, simply open the file to visualize the plot.

In this step, you'll create a new stored procedure, *PlotInOutputFiles*, that demonstrates how to use R plotting functions to create histograms, scatterplots, and more in .JPG and .PDF format. The graphics files are saved to local files; that is, in a folder on the SQL Server instance.

To create the stored procedure PlotInOutputFiles

1. In SQL Server Management Studio, open a new query window, and paste in the following Transact-SQL statement.

```
CREATE PROCEDURE [dbo].[PlotInOutputFiles]
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`.

2. Run the statement to create the stored procedure.

To generate the graphics files

1. In SQL Server Management Studio, run the following SQL query:

   ```
   EXEC PlotInOutputFiles
   ```

   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

2. Open the destination folder and review the files that were created by the R code in the stored procedure. (The numbers in the file names are randomly generated.)

   - rHistogram_Tipped_*nnnn*.jpg: Shows the number of trips that got a tip (1) vs. the trips that got no tip (0). This histogram is much like the one you generated in the previous step.

   - rHistograms_Tip_and_Fare_Amount_*nnnn*.pdf: Shows the distribution of values in the tip_amount and fare_amount columns.

     ```
      ![histogram showing tip_amount and fare_amount](media/rsql-devtut-tipamtfareamt.PNG "histogram
     showing tip_amount and fare_amount")
     ```

   - rXYPlots_Tip_vs_Fare_Amount_*nnnn*.pdf: A scatterplot with the fare amount on the x-axis and the tip amount on the y-axis.

     ```
      ![tip amount plotted over fare amount](media/rsql-devtut-tipamtbyfareamt.PNG "tip amount plotted
     over fare amount")
     ```

3. 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.

Next Step

Step 4: Create Data Features using T-SQL

Previous Step

Step 2: Import Data to SQL Server using PowerShell

See Also

In-Database Advanced Analytics for SQL Developers (Tutorial)
SQL Server R Services Tutorials

Step 4: Create Data Features using T-SQL (In-Database Advanced Analytics Tutorial)

4/19/2017 • 3 min to read • Edit Online

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 features from the raw data is a critical step in advanced analytics modeling.

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.

To 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
```

- 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*.

To 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 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.

In the next step, you'll learn how to use these data features to train a machine learning model using R.

Next Step

Step 5: Train and Save a Model using T-SQL

Previous Step

Step 3: Explore and Visualize the Data

See Also

In-Database Advanced Analytics for SQL Developers (Tutorial)

SQL Server R Services Tutorials

Step 5: Train and Save a Model using T-SQL

4/29/2017 • 3 min to read • Edit Online

In this step, you'll learn how to train a machine learning model by using R. The R packages are already installed with R Services (In-Database) so you can call the algorithm from a stored procedure. You'll train the model using the data features you just created, and then save the trained model in a SQL Server table.

Building an R Model using Stored Procedures

All calls to the R runtime that is installed with R Services (In-Database) are done by using the system stored procedure, sp_execute_external_script. However, if you need to retrain a model, it is probably easier to encapsulate the call to sp_execute_exernal_script in another stored procedure.

In this section, you'll create a stored procedure that can be used to build a model using the data you just prepared. This stored procedure defines the input data and uses an R package to create a logistic regression model.

To create the stored procedure

1. In Management Studio, open a new Query window and run the following statement to create the stored procedure *TrainTipPredictionModel*.

   Note that, because the stored procedure already includes a definition of the input data, you don't need to provide an input query.

```
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(serialize(logitObj, NULL)));
',
                                  @input_data_1 = @inquery,
                                  @output_data_1_name = N'trained_model'
  ;

END
GO
```

   This stored procedure performs these activities as part of model training:

- 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.

To create the R model using the stored procedure

1. In Management Studio, run this statement.

   ```
   EXEC TrainTipPredictionModel
   ```

   Processing of the data and fitting the model might take a while. Messages that would be piped to R's **stdout** stream are displayed in the **Messages** window of Management Studio. For example:

STDOUT message(s) from external script: Rows Read: 1193025, Total Rows Processed: 1193025, Total Chunk Time: 0.093 seconds

Successive chunks of data are read and processed. You might also see messages specific to the individual function, `rxLogit`, indicating the variable and test metrics.

1. 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*.

   model 0x580A00000002000302020....

In the next step you'll use the trained model to create predictions.

Next Step

Step 6: Operationalize the Model

Previous Step

Step 4: Create Data Features using T-SQL

See Also

In-Database Advanced Analytics for SQL Developers (Tutorial) SQL Server R Services Tutorials

Step 6: Operationalize the Model (In-Database Advanced Analytics Tutorial)

4/19/2017 • 6 min to read • Edit Online

In this step, you'll learn to *operationalize* the model using a stored procedure. This stored procedure can be called directly by other applications, to make predictions on new observations.

In this step, you'll learn two methods for calling an R model from a stored procedure:

- **Batch scoring mode**: Use a SELECT query to provide multiple rows of data. 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 that will be scored 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 R, 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 a tip (of any amount) will be given.

Batch scoring using a SELECT query

Now let's see how batch scoring works.

1. Let's start by getting a smaller set of input data to work with.

```
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
```

This query creates a "top 10" list of trips with passenger count and other features needed to make a prediction.

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

1 276 0.7 2013-06-27 06:53:04.000 0.4478814362

1 282 0.7 2013-02-21 07:59:54.000 0.5340645749

1 260 0.7 2013-03-27 15:40:49.000 0.5513900727

1 230 0.7 2013-02-05 09:47:59.000 0.5161578519

1 250 0.7 2013-05-08 14:35:51.000 0.5626440915

1 280 0.7 2013-05-11 12:22:01.000 0.5598517959

1 308 0.7 2013-04-10 08:06:00.000 0.4478814362

You'll use this query as input to the stored procedure, *PredictTipBatchMode*, which was provided as part of the download.

1. First, take a minute to review the code of the stored procedure *PredictTipBatchMode* in Management Studio.

```
/****** Object:  StoredProcedure [dbo].[PredictTipBatchMode]  ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[PredictTipBatchMode] @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
```

2. To create predictions, you'll provide the query text in a variable and pass it as a parameter to the stored procedure, using a Transact-SQL statement like this.

```
-- Specify input query

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 stored procedure for scoring
EXEC [dbo].[PredictTip] @inquery = @query_string;
```

3. The stored procedure returns a series of values representing the prediction for each of the "top 10 trips". Looking back at the input values, all of the "top 10 trips" are single-passenger trips with a relatively short trip distance. Based on the data, the driver is very unlikely to get a tip on such trips.

   Rather than returning just the yes-tip/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.

Score Individual Rows

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'll learn how to create single predictions using a stored procedure.

1. Take a minute to review the code of the stored procedure *PredictTipSingleMode*, which was 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. For more information, see Working with R Data Types.

- 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, typing parameters for each of the feature columns.

   ```
   EXEC [dbo].[PredictTipSingleMode] 1, 2.5, 631, 40.763958,-73.973373, 40.782139,-73.977303
   ```

   The values are for these feature columns , in order:

   - *trip_time_in_secs*
   - *pickup_latitude*
   - *pickup_longitude*
   - *dropoff_latitude*
   - *dropoff_longitude*

3. The results indicate that the probability of getting a tip is very low on these top 10 trips, all of which are single-passenger trips over a relatively short distance.

Conclusions

In this tutorial, you've learned how to work with R code embedded in stored procedures. 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 Step

Step 4: Create Data Features using T-SQL

See Also

In-Database Advanced Analytics for SQL Developers (Tutorial)
SQL Server R Services Tutorials

Data Science Scenarios and Solution Templates

4/19/2017 • 4 min to read • Edit Online

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, and includes sample data, R code (Microsoft R Server) and SQL stored procedures. The tasks in each template extend from data preparation and feature engineering to model training and scoring. The code can be run in an R IDE, with computations done in SQL Server, or by using a SQL client tool such as SQL Server management Studio.

You can use these templates to learn how R Services (In-Database) works, and build and deploy your own solution by customizing he template to fit your own scenario.

For download and setup instructions, see How to Use the Templates at the end of this topic.

Fraud Detection

Online Fraud Detection Template (SQL Server R Services)

One of the important tasks for online business is to detect fraudulent transactions, and to identify the transactions made by stolen payment instruments or credentials, in order to reduce charge back losses. 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'll learn how to use data from online purchase transactions to identify likely fraud. This methodology is one that you can easily apply to fraud detection in other domains.

In this template, you'll learn how to use data from online purchase transactions to identify likely fraud. 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.

Customer Churn

Customer Churn Prediction Template (SQL Server R Services)

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.

This template get you started with churn prevention by formulating 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)

The goal of "data-driven" predictive maintenance is 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 particularly important for applications that rely on distributed data or sensors, as exemplified by the Internet of Things (IoT).

This template focuses on answering the question of “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, sensor measurements and so forth, 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 (number of days).

  The **multi-class classification model** predicts whether a particular engine will fail, and if it will fail, 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

This template demonstrates how to use SQL Server R Services to predict demand for electricity. 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.

See Also

SQL Server R Services Tutorials
Announcing the Templates in Azure ML
New Predictive Maintenance Template