Contents

[Intro 1](#_Toc434520375)

[Key Scenarios Description 1](#_Toc434520376)

[Setup Instructions 3](#_Toc434520377)

[Scenario: Operationalizing R Code via T-SQL (a.k.a. Developer Scenario) 3](#_Toc434520378)

[1. Prepare a Server 3](#_Toc434520379)

[2. Install SQL Server 2016 CTP3 with SQL Server R Services 3](#_Toc434520380)

[3. Sanity test that the scenario is working 3](#_Toc434520381)

[Scenario: Data Exploration and Predictive Modeling (a.k.a. Data Scientist Scenario) 4](#_Toc434520382)

[4. Configure & Sanity test that the scenario is working 4](#_Toc434520383)

[Initial samples and guides 5](#_Toc434520384)

[Data Exploration and Predictive Modeling 5](#_Toc434520385)

[Operationalizing R Code using T-SQL 6](#_Toc434520386)

[Pre-Requisites 6](#_Toc434520387)

[Install packages on SQL Server machine 6](#_Toc434520388)

[Sample Database Setup 6](#_Toc434520389)

[Operationalizing a Model using iris dataset and Predict function 6](#_Toc434520390)

[Generating plots for iris dataset from a stored procedure 6](#_Toc434520391)

[Operationalizing a Model using iris dataset and rxPredict function 7](#_Toc434520392)

[Fraud Detection Sample using Benford Law 7](#_Toc434520393)

[Known Issues 7](#_Toc434520394)

# Intro

This document is meant to help you get started with SQL Server R Services (in-database). This is a new feature being introduced in SQL Server 2016 CTP3. The document contains:

* [Key scenarios description](#_Key_Scenario_Description)
* [Setup Instructions](#_Setup_Instructions)
* [References to initial samples and guides](#_Initial_samples_and)
* [List of known issues](#_Known_Issues)

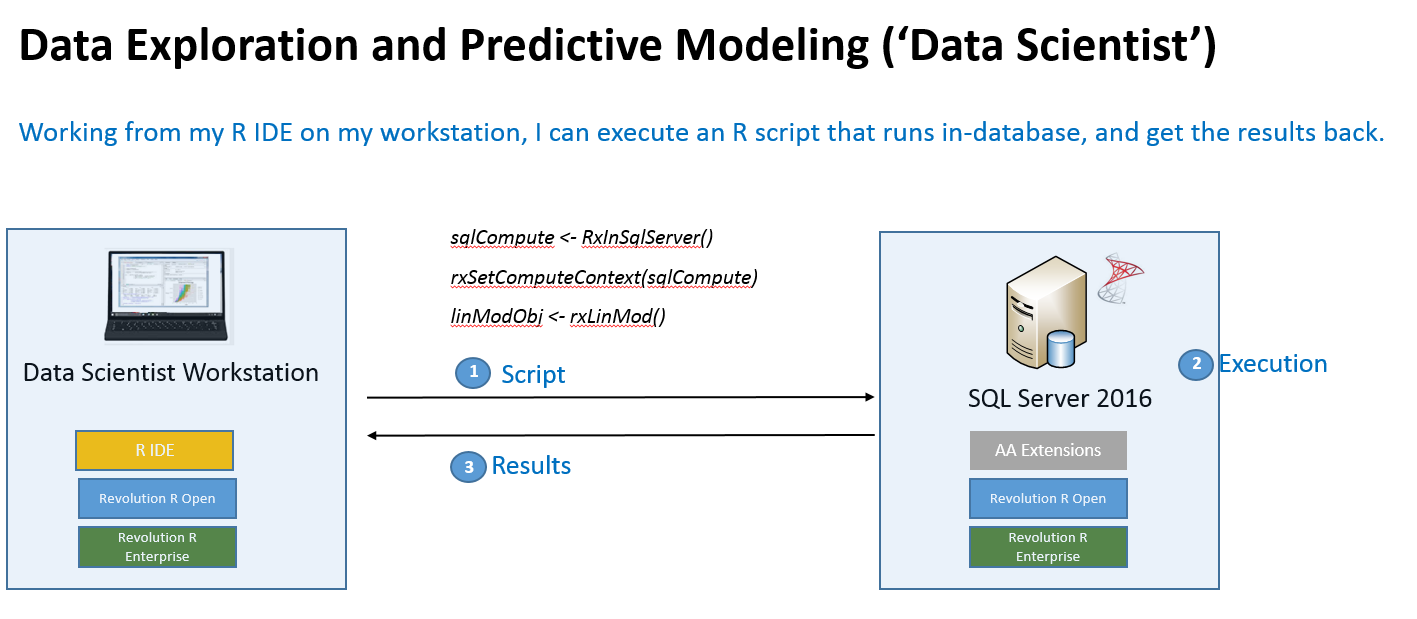
# Key Scenarios Description

SQL Server R Services is all about helping customers embrace the highly popular open source R language in their business. R is the most popular programming language for advanced analytics. You can use it to analyze data, uncover patterns and trends and build predictive models. It offers an incredibly rich set of packages and a vibrant and fast-growing developer community. At the same time, embracing R in an enterprise setting presents certain challenges, especially as the volume of data rises and with the switch from modeling to production environments. Microsoft SQL Server R Services with in-database analytics helps customers embrace this technology by supporting several scenarios. Two of the key scenarios are:

Data Exploration and Predictive Modeling with R over SQL Server data:

This step is typically performed by R Developers / Data Scientists. The benefits include:

* **Familiar user interface:** youget to workwith your R IDE of choice.
* **In-DB processing**: you can execute R code and have the computations take place in-database (on the SQL Server machine). This eliminates the need of having to move data from the DB to an external machine running R.
* **Performance and scale:** by using the RevoScaleR package APIs, you are no longer restricted by R’s single thread and memory-bound architecture. You can now work with large datasets and multi-threaded, multi-core, multi-process computations.
* **Code portability:** the same R code you run against SQL Server data can be used against other databases such as Hadoop.

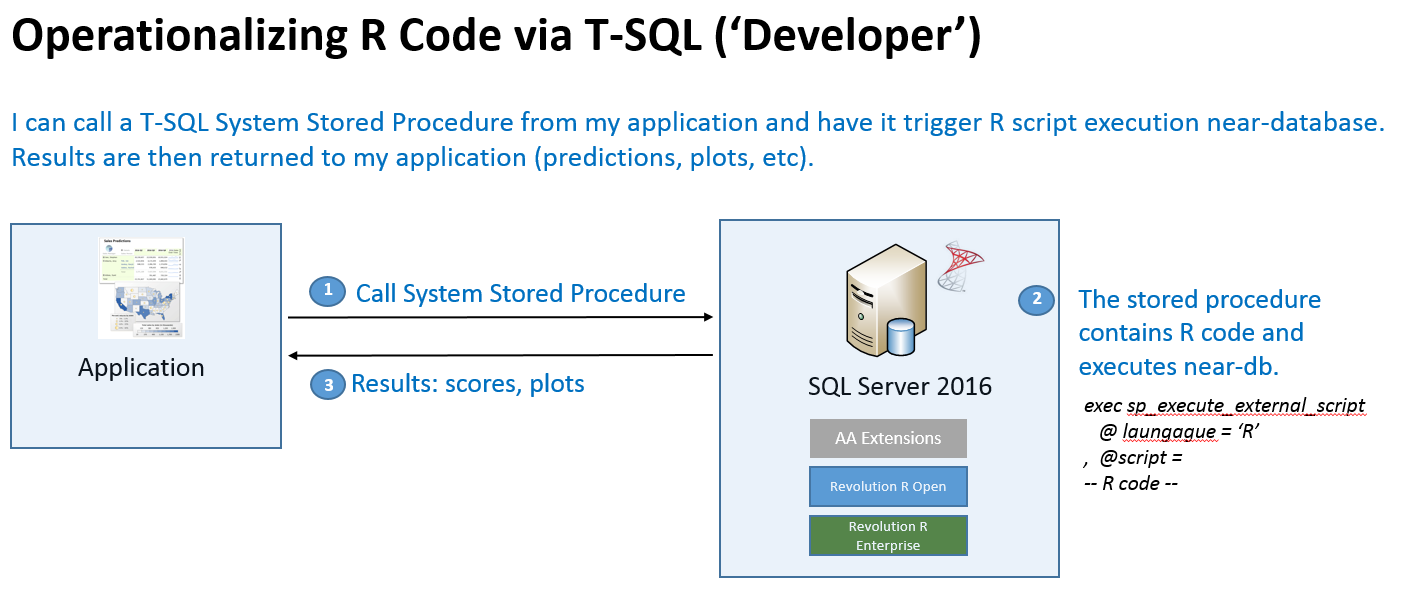


See more details about this scenario in the [samples section](#_Data_Exploration_and).

Operationalizing your R code via T-SQL:

This step is typically performed by Application Developers / SQL Developers. The benefits include:

* **T-SQL operationalization**: you can invoke R scripts by embedding them in system stored procedures and have the computation execute in-DB. This is useful when you want to score against a predictive model in production. You can also execute R scripts and return plots and predictions and embed them in your application, for instance in SQL Server Reporting Services reports.
* **Performance and scale:** system stored procedures that invoke the RevoScaleR package APIs can operate on large datasets and benefit from multi-threaded, multi-core, multi-process in-DB computations.
* **Familiar user interface:** you get towork with the familiar SQL Server Management Studio interface.



See more details about this scenario in the [samples section](#_Operationalizing_R_Code).

# Setup Instructions

## 

## Scenario: Operationalizing R Code via T-SQL (a.k.a. Developer Scenario)

You will need to install and configure SQL Server 2016 CTP3, Revolution R Open (RRO) and Revolution E Enterprise (RRE) on the server.

### 1. Prepare a Server

You would want a solid machine that has Internet connectivity and allows you to connect from a client machine (data scientist workstation). We recommend spinning-up a VM and not installing the bits on your personal workstation.

### Install SQL Server 2016 CTP3 with SQL Server R Services

Follow instructions from below for setting up server:

<https://msdn.microsoft.com/en-US/library/mt604883.aspx>

### Sanity test that the scenario is working

Open SQL Server Management Studio and execute the following query. It should return ‘1’.

EXECUTE sp\_execute\_external\_script

@language = N'R'

, @script = N'OutputDataSet <- InputDataSet'

, @input\_data\_1 = N'SELECT 1 as Col'

WITH RESULT SETS ((col int not null));

go

## Scenario: Data Exploration and Predictive Modeling (a.k.a. Data Scientist Scenario)

To enable this scenario, you need to go through all the setup and configuration steps listed for the Operationalization scenario (steps 1-6 above), and then setup a client machine and perform some additional configurations.

Follow the instructions for setting up a data science workstation from below link:

<https://msdn.microsoft.com/en-US/library/mt604883.aspx>

### Configure & Sanity test that the scenario is working

#### Import test data into a new DB

The file “ImportAirlineDB.sql” contains code to create test database. Run the script from SQL Server Management Studio. (you can just copy paste it into a new query, or use CTRL+O to browse for the query file, pick it and execute it)

#### Create a SQL login

Run the following query from SQL Server Management Studio. It creates a SQL login that has permission to query the tables in the RevoTestDB. It also adds this user to the db\_rrerole that allows executing external stored procedures. Replace MyUser and MyPassword in the query below with actual values of your choice (note: replace only the highlighted parts, keep the []).

USE [RevoTestDB]

  GO

  CREATE LOGIN [MyUser] WITH PASSWORD= ‘MyPassword’, CHECK\_EXPIRATION=OFF, CHECK\_POLICY=OFF;

  CREATE USER [MyUser] FOR LOGIN [MyUser] WITH DEFAULT\_SCHEMA=[db\_datareader]

  ALTER ROLE [db\_datareader] ADD MEMBER [MyUser]

USE [master]

  GO

   CREATE USER [MyUser] FOR LOGIN [MyUser] WITH DEFAULT\_SCHEMA=[db\_rrerole]

   ALTER ROLE [db\_rrerole] ADD MEMBER [MyUser]

#### (Optional) Verify the SQL login creation

* In SQL Server Management Studio under security -> logins, double click MyUser and open User Mapping.
  1. Make sure the user is added to the db\_rrerole in master database
  2. Make sure the user is added to the db\_datareader in the RevoTestDB

#### Run the test R script

Launch Revolution R Enterprise on your client (this will open the IDE) and run the below script after replacing the <> to whatever you created in the previous step. The vmhostname stands for the SQL Server host name. If it’s not on the same domain as the client you should use the FQDN.

connectionString <- "Driver=SQL Server;Server=vmhostname;Database=RevoTestDB;Uid=MyUser;Pwd=MyPassword"

airData <- RxSqlServerData(

connectionString = connectionString,

table = "AirlineDemoSmall",

colInfo = list( ArrDelay = list(type = "integer"), DayOfWeek = list(type = "factor", levels = c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")))

)

cc <- RxInSqlServer(connectionString = connectionString, autoCleanup = FALSE, consoleOutput = TRUE)

rxSetComputeContext(cc)

rxSummary(~ArrDelay + DayOfWeek, data = airData)

# Initial samples and guides

## Data Exploration and Predictive Modeling

The data scientist can choose to analyze the data in-database or they can pull data from SQL Server and analyze it on their client machine (or a separate server). The former has advantages by removing the need to move data around.

Either way, we encourage customers to work with the RevoScaleR package and APIs. This package contains a set of common functions and algorithms that were designed for performance and scale, overcoming R limitations of single-threaded execution and memory bound datasets. The package also allows pushing the compute to the SQL Server machine and thus avoiding data movements over the network. This is achieved by using the SQL Server ‘compute context’.

If you are already familiar with R and wish to understand how to leverage RevoScaleR for SQL Server, you can follow the RevoScaleR\_SqlServer\_Getting\_Started.pdf found under: “%ProgramFiles%\RRO\RRO-3.2.2-for-RRE-7.5.0\R-3.2.2\library\RevoScaleR\doc”

## Operationalizing R Code using T-SQL

#### T\_SQL Reference

For SQL Server 2016 CTP3, we support ad-hoc execution of R scripts via a new system stored procedure. This stored procedure will support pushing data from a single SELECT statement & multiple input parameters to the R side and return a single data frame as output from the R side. The documentation for the new system stored procedure sp\_execute\_external\_script can be found below:

<https://msdn.microsoft.com/en-US/library/mt604368.aspx>

## Pre-Requisites

[SQL Server Management Studio](https://msdn.microsoft.com/library/mt238290.aspx) to execute the SQL scripts

[SQL Server 2016 Report Builder](http://www.microsoft.com/en-us/download/details.aspx?id=49162) to view the sample reports

[R Studio Desktop](https://www.rstudio.com/products/rstudio/#Desktop) or another R IDE to install packages

## Install packages on SQL Server machine

The R script “install\_packages.r” installs packages required for the samples. To install the packages on the SQL Server machine, please follow the steps below:

* Start your R IDE as administrator (*Tip: Right-click on the IDE shortcut & use “Run as Administrator”*)
* Run the script “install\_packages.r” from your R IDE. This will provision the packages for all users of the R environment

Alternatively, you can run the R script “install\_packages.r” from R.exe running from an elevated command prompt

## Sample Database Setup

The file “sample\_db\_setup.sql” contains sample code to create a test database with low privileged user “rdemo” that will be used in various reports / R scripts. Run this script first from SQL Server Management Studio to create the sample database required for the scenarios

## Operationalizing a Model using iris dataset and Predict function

The file “iris\_demo.sql” contains stored procedures that demonstrate how to train a model from SQL Server, store the model in a database table & use model to predict. Run this script from SQL Server Management Studio to create the stored procedures.

The stored procedure “generate\_iris\_model” uses Naïve Bayes algorithm in e1071 package to train a model based on the iris dataset. The trained model is then stored into a SQL table “iris\_models”.

The stored procedure “predict\_species” uses the predict function in e1071 package to predict species attribute of each flower based on the iris dataset.

## Generating plots for iris dataset from a stored procedure

The file “iris\_demo.sql” contains two stored procedures “get\_iris\_plot1” and “get\_iris\_plot2” that demonstrates how to generate plot from T-SQL using ggplot2 package. Run this script from SQL Server Management Studio to create the stored procedures required for this scenario.

The plots can be viewed by opening the “iris\_plot.rdl” file using [SQL Server 2016 Report Builder](http://www.microsoft.com/en-us/download/details.aspx?id=49162). The report assumes that you installed a default instance of SQL Server on your machine. For instructions on how to modify the data source settings refer [here](https://technet.microsoft.com/en-us/library/dd207039(v=sql.110).aspx)

## Operationalizing a Model using iris dataset and rxPredict function

The file “iris\_rx\_demo.sql” contains stored procedures that demonstrate how to train a model from SQL Server, store the model in a database table & use model to predict using RevoScaleR functions. Run this script from SSMS to create the stored procedures required for this scenario.

The stored procedure “generate\_iris\_rx\_model” uses rxLinMod function from RevoScaleR package to train a model based on the iris dataset. The trained model is then stored into a SQL table “iris\_rx\_models”.

The stored procedure “predict\_species\_sepal\_length” uses the rxPredict function in RevoScaleR package to predict sepal length attribute of each flower species based on the iris dataset.

## Fraud Detection Sample using Benford Law

The file “benford\_setup.sql” creates tables used for the fraud detection sample. Run this script first to setup the tables required for the sample.

The file “benford\_demo.sql” contains the stored procedures that will be used to demo Benford’s law. For more information on Benford’s law see [here](https://en.wikipedia.org/wiki/Benford%27s_law). Run this script from SQL Server Management Studio to create the stored procedures required for this scenario.

The stored procedure “getPotentialFraudulentVendors” uses Benford’s law to generate a list of fraudulent vendors from the “Fraud” table. The stored procedure takes a parameter @threshold that can be used to filter out rows that may not qualify as fraud.

The stored procedure “getVendorInvoiceDigits” uses ggplot2 package to generate a plot that shows the actual distribution of the first digits of invoice amounts along with the expected distribution according to Benford’s law. The plot can be used to visualize the actual vs expected results and determine if any fraud was involved. The plot is returned as a JPG file & the stored procedure can be called from a report to see the output.

The stored procedure “getVendorInvoiceDigitsPlots” demonstrates how to integrate fraud detection & visualization of the results. In this stored procedure, we will generate the list of fraudulent vendors first and generate plot for each vendor that shows the actual vs expected distribution of the first digits of invoice amounts.

The generated plots are stored in a table that can be consumed by the report “benford\_FraudDetection.rdl”. The report can be viewed by using [SQL Server 2016 Report Builder](http://www.microsoft.com/en-us/download/details.aspx?id=49162). The report assumes that you installed a default instance of SQL Server on your machine. For instructions on how to modify the data source settings refer [here](https://technet.microsoft.com/en-us/library/dd207039(v=sql.110).aspx)

# Known Issues

|  |  |
| --- | --- |
| Topic | Description |
| Performance will improve by RTM | Performance is not yet optimal in CTP3. ScaleR functions don't support multi-process parallelism. Pushing data in parallel from SQL to R and Batch mode execution of queries is also expected in RTM. |
| No support for multiple result sets | No support for multiple result sets in CTP3. A mitigation can be issuing multiple calls or writing data to disk via ODBC and have the app read it later. |
| No support for output parameters | Same as above |
| SQL authentication mandatory from client | Working from the Data scientist client is only supported with SQL authentication until we will move to a new architecture post CTP3. |
| Resource governance | While using SQL Server resource pools is not available yet in CTP3, we have placed a cap of 20% on the max resource usage of the machine resources. This value is configurable. |
| Running side by side with additional RRE versions | Installing RRE side by side with non-CTP3 RRE versions is not supported. |
| RevoScaleR rxExec not working in CTP3 | The rxExec function is not-usable until we provide support for parallelism across processes. |
| String corruptions | Roundtrip of strings from SQL -> R -> SQL can result in corruptions. This is due to difference in encodings between R & SQL Server and collation/language. |
| Datetime values incomplete | Datetime values can lose milliseconds values due to difference in precision |
| Datatype inconsistencies | Many datatype ranges are different between SQL Server & R. As a result, sending such values from R can result in NA values on the R side. Similarly, sending values from R that are unsupported in SQL Server can result in NULL values (ex: +Inf, -Inf, NaN float values). |
| Outbound network access is blocked for R sessions | If Windows firewall is enabled on the SQL Server machine, by default we block outbound network access from R processes. As a result, any operation that accesses external network resources from R script like file shares, web sites, etc. will fail |