[[1]](#footnote-2)

Proposal for Term Project

Title: SQL Server 2016 R Services Deep-Dive

(MSDS 7330, Fall 2016)

Christopher Boomhower, Alex Frye, and Lindsay Vitovsky

*Abstract*—This project involves investigating the usefulness of a new functionality released by Microsoft that supports integration between R and SQL Server 2016. Movement of data from a database to separate IDEs becomes painful with large datasets and causes potential security risks Through R Services in SQL Server 2016, the computational power of R is combined with the ease of storage and access of a Database Management System (DBMS). We seek to explore this new capability, and determine if it can successfully address the challenge of performing research on large data sets inside of separate analysis and presentation tools / IDEs.

# INTRODUCTION

P

erhaps the two most central “thirsts” of data science are to 1) develop a more accurate understanding of a population, and 2) be able to make meaningful predictions about those populations. However, with the term “big data” approaching almost hyperbolic levels, the integrity of addressing these two tasks can easily become lost in the web of various software libraries, hardware capabilities, and information security challenges. As former Chief Information Officer for Google Douglas Merrill, Ph.D. wrote in 2012 for his article “R is Not Enough for ‘Big Data’”,

*“But who cares about how much data you have? With too little data, you won’t be able to make any conclusions you trust. With loads of data you will find relationships that aren’t real.”[[2]](#endnote-2)*

This balance of trying to compute meaningful interpretations while being able to handle increasingly large amounts of data is the crux of our paper. Our team was intrigued by the impact technological horsepower can have on the judiciousness of data interpretation. Put simply, one can have a beautifully written piece of software that can handle complicated calculations and produce powerful visualization, but if it cannot produce this within an acceptable amount of time and with a satisfactory amount of data, what does it matter?

Our team will be researching and exploring an integration of one of the most popular statistical software tools, R, and the standard in Relationship Database Management Systems, SQL Server. There are three core challenges to research and explore during this project: 1) Data Movement, 2) Scale and Performance, and 3) Operationalizing to Production System(s). We aim to present our findings in a way that will give the reader an accurate perception of the benefits this integration provides to data scientists, and where it fits.

# BACKGROUND

Why R?

Certainly, a major challenge for data scientists is how to prudently and reliably make interpretations that actually mean something to their audience. It is not just human judgment that contributes to meaningful analysis; one needs capable tools and computing power to promote ethical and consequential data evaluation.

On its own, R remains a heavily used software program to produce statistical models, graphics, and calculations. Its wealth of libraries, maintained by a global community of users, give researchers many options for how to best clean, process, and present data. Despite the increasing popularity of other languages, namely Python, R continues to be a powerful tool for data scientists.

Given that R is processed in the main memory, however, it is not conducive to big data analysis. R is limited to the amount of RAM on the user’s machine[[3]](#endnote-3), and while this offers computational efficiency and speed, one can only push R so far before they find themselves waiting for undesirable lengths of time or perhaps even crashing one’s IDE.

Even if R could magically handle large amounts of data within a user’s available RAM, you are then met with potential privacy concerns. Individual subject data might need to remain where it is stored for security reasons. R does not have the ability on its own to process data stored elsewhere without importing the data to be analyzed.

SQL Server 2016 R Services

SQL Server 2016 features a new integration for enterprise level data analytics, R Services. Using one of the most popular data analysis tools with one of the standards in data storage and management could benefit data scientists, database developers, database administrators, and data engineers. Besides the three components we will be researching for this paper (data movement, scale and performance, and operationalization), the ability to use an open source software, one which does not require an extensive amount of technical knowledge, means this integration could possibly create an even broader audience than for either software package on its own.

R Services utilizes the *RevoScaleR* package in R for scalable, high performance data movement and analysis. It is included in the SQL Server 2016 Microsoft R Services download, and demonstrated later in section III, data movement. *RevoScaleR* is meant to help R overcome the challenge of slow data computation due to large data sets, and has been used in multiple APIs to improve the performance of R. It was developed by Revolution Analytics (founded in 2007) and released in 2010. Microsoft acquired Revolution Analytics in 2015.

Software / hardware requirements:

To complete this term paper, we will require a few particular resources.

* A Windows Computer (Version 8 or higher)
* Reliable Internet Connection
* SQL Server 2016 installation with SQL Server R Services enabled
* db\_datareader, db\_datawriter privileges to the RServicesDemo Database [[4]](#endnote-4)
* Microsoft R Server or Microsoft R Client for *RevoScaleR* package installation
* All R packages installed in SQL server instance

# Data movement

One of the most prevalent challenges associated with DBMSs is that of system interaction, or more simply – data movement. It goes without saying that if data is stored in an active DBMS, it will likely be queried, manipulated, or analyzed in some fashion at some point in time. R packages such as RODBC, RMySQL, ROracle, and RJDBC address the need of data transfer between an R session and DBMS. However, they do nothing more than function as interfaces to the database. This means a researcher must read from the database, save locally, implement computations, and then write back to the database when complete. This equates to latency, duplicate data, and inefficient use of resources. In contrast, being able to utilize the statistical power of R within the DBMS environment itself would directly address these data movement concerns.

There are, of course, other basic functions outside the realm of R that are meant to address the challenges associated with data movement. Perhaps one of the simplest solutions involves serial loading of the data in combination with incremental transfer. This involves copying all changed records within the database based on a particular time point. Often, such data transfer may be setup to occur automatically; however, such scheduling may prove counterproductive since data movement can be rather time-intensive and additional updates may be applied to the records before transfer is complete. At that point, researchers are compelled to work with outdated data. One solution to such problems might be to expand the functionality of MySQL’s binary log, for example, such that data change identification may be leveraged by analytics applications as well, resulting in a seamless record transfer for immediate researcher access upon each change incident.

Though other solutions remain, perhaps the best solution for many situations is an R solution after all. The *RevoScaleR* package is a library included with SQL Server 2016’s Microsoft R Services. Utilizing R Services via this package marries the size, structure, and efficiency of a DBMS with the sophisticated, high performance analytics capabilities of R. How is the accomplished? – By performing all computations at the same location as the data residing in the SQL Server 2016 database. As data storage/management becomes more integrated with data analysis via this approach, data science roles become more centralized, efficient, and cost-effective. These benefits culminate into a sustainable workflow within an organization, and this enhancement is founded upon data movement reduction and, therefore, increased speed.

In order to test our theory of increased speed due to reduced need for data movement, we performed a simple experiment using a dummy data set containing 10 million records. These records were first stored within a SQL Server 2016 database. Next, all 10 million records were loaded into the R environment via two approaches and their completion times recorded. The first approach represents the traditional means of data analysis by pulling the data into an external, running instance of R, and the second represents the noted R Services approach of loading the data into an R instance running within the data’s database location. Loading a data set of this magnitude from a DBMS into an independent R process took 19 minutes and 52 seconds on our data science workstation.[[5]](#endnote-5) Loading the same data from the same database, on the same workstation, into a R Services instance running at the data’s location within the database, completed in 1 minute and 20 seconds![[6]](#endnote-6) Clearly, the R Services method using *RevoScaleR* takes the win for speed, and in turn, efficiency. Bear in mind that once the data is processed and analyzed it may very well require write-back to the database, extending the duration of data movement even further.

Further establishing Microsoft R Services as an optimal solution to the data movement dilemma, installing Microsoft’s Revolution R Enterprise platform onto both the computer hosting the SQL Server 2016 database and onto one’s data science workstation allows for data scientists to interact freely with their remote database all from within their local IDE. Though this concept may sound familiar, differentiation lies in the ability to deploy all R scripts, developed on the data science workstation, into the SQL Server database environment as previously discussed. As depicted in Figure 1, R communication is established such that scripts are passed to the server for execution after which results are returned to the data science workstation environment. In this way, development may proceed in the same familiar setting with which many data scientists are familiar while harnessing the high-performance, fast, scalable, parallel architecture of *RevoScaleR* in the SQL Server 2016 environment.

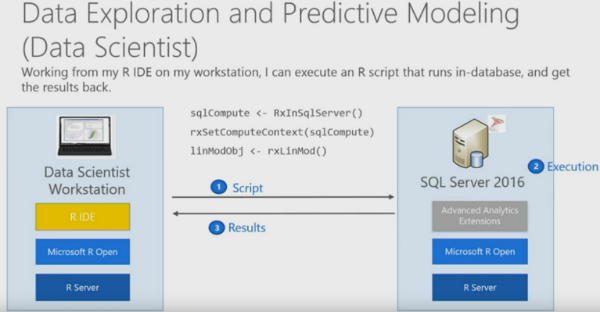


Figure 1. Revolution R Services Functionality Example (Courtesy of Elharrar, Microsoft Cloud Platform)

Of course, while Microsoft R Services addresses many needs in terms of data movement, several short-comings still linger. For one, R may not be fit for all analytics solutions. If a given scenario requires utilization of another statistical software package, data movement is still a problem. Furthermore, beyond the scope of a base analytics application as discussed thus far, the need for more efficient data transfer is growing at a rapid rate, day-after-day. This growth is accelerating as the standard for data set sizes increases and as society begins to rely more heavily on AI technologies which demand virtually instant data acquisition and processing. While R Services is a valid solution to many data movement setbacks, there is still much need for improvement to address the aforementioned scenarios.

# scale / performance

Of equal challenge to modern big data analysis is that of maximizing scalability and performance. On its own, the performance and speed of R is somewhat constrained as it depends heavily on the environment in which it is being used. R holds all objects in virtual memory, and depending on one's operating system, it might have limitations placed on it to the amount of resources that can be used at one time. When trying to process large datasets, or use a library with poorly written or cumbersome code, one can easily find himself/herself receiving error messages with R unable to obtain enough memory to process the needed computation.

SQL Server has been optimized over many years and product releases to continuously improve on parallel query plans. In small queries, a likely query plan will utilize a single “worker” to provide desired results. In Figure 2 below, an example query plan for a small table record count is provided. The plan will scan the table index, aggregate the count, and provide outputted results.



Figure 2. Single Threaded Execution Plan  
(Courtesy of Paul White, Simple-Talk)

For larger queries, the SQL optimizer may choose to utilize several “workers” to execute the process as is displayed in Figure 3 for a large table record count. The optimizer will utilize a feature called the Parallel Page Supplier to distribute load among the workers. This distribution is based on demand and availability of each worker, and ensures that no two workers receive the same records to process. The workers will complete their index scan and aggregate count as was displayed previously and feed into Gather Stream Operator. SQL Server would then aggregate the results from each worker, providing final outputted results.

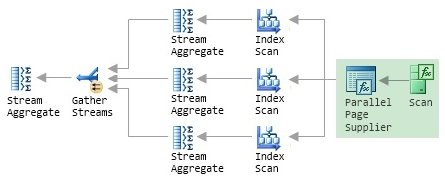


Figure 3. Parallel Execution Plan  
(Courtesy of Paul White, Simple-Talk)

The SQL Server Engine is built in a way to automatically build query plans to benefit from multi-core servers. The ability to split processing to multiple workers provides tremendous performance improvements to queries and allows a query to scale as data volume increases.

Some research has been conducted and attempts made to address the computational deficiencies of implementing R scripts at a larger scale, but these venues of research do little to address key challenges in terms of DBMS interaction. For example, because R is designed to compute within main memory (RAM), which improves computational efficiency and speed locally, and R core programming is single-threaded only, researchers are limited by data scale and single-processor speed. The *parallel* package, released with R version 2.14.0, addresses these issues, but does not address the concerns of data movement, operationalization, and performance within the DBMS environment. The *RevoScaleR* package is a similar package that expands upon these principles of parallel computing by also using the *.xdf* data file format to enhance processing speeds of data blocks. Fortunately, since *RevoScaleR* is an integrated component of Microsoft R Services, the two address the need for parallel processing in tandem.

At a high-level, R Services addresses scalability by relying on the SQL Server machine’s available cores to perform parallel data processing. Integration of the *RevoScaleR* package optimizes such parallel processes by managing all parallel processes automatically. Parallel processing support is also made available for situations where the *RevoScaleR* package may not meet an application’s requirements or in cases where legacy scripts are adopted into a functioning R Services solution. This is accomplished by setting the *@parallel* parameter to 1 as part of one’s external script procedure (external with respect to the data science workstation; internal with respect to the database). In both parallelism scenarios, the SQL Server creates multiple parallel processes based on the server’s maximum degree of parallelism (MAXDOP) definition. Parallel processing is managed automatically when *RevoScaleR* is used; however, options are available for additional user control – the number of parallel tasks may be set by the data scientist when *RevoScaleR* is not incorporated (via the *numtasks* designator) and the data batch size may be specified to determine how many rows to process at a time (via the *rowsPerRead* designator). Fifty-thousand rows are processed at a time by default but this number may be increased for server machines containing sufficient memory.

As the demand for increased parallelism grows larger, the need for solutions such as *RevoScaleR* grows as well. While this tool does address the challenges associated with scalability/performance and even data movement head-on, its support for operationalization challenges is lacking. As such, further discussion is provided toward this latter topic in the section below.

# operationalize to production systems

Another challenge for organizations today is streamlining big data projects into production systems. Data scientists create and present very complex algorithms and/or models for their business to provide insights towards important decisions. These algorithms are typically created on the data scientist’s workstation, connecting to various data sources – be it DBMS systems, CSV Files, or data from the web. Once these algorithms are complete, the data scientist may run an ad-hoc RMarkdown report against production data to present to business stakeholders (See Figure 4).

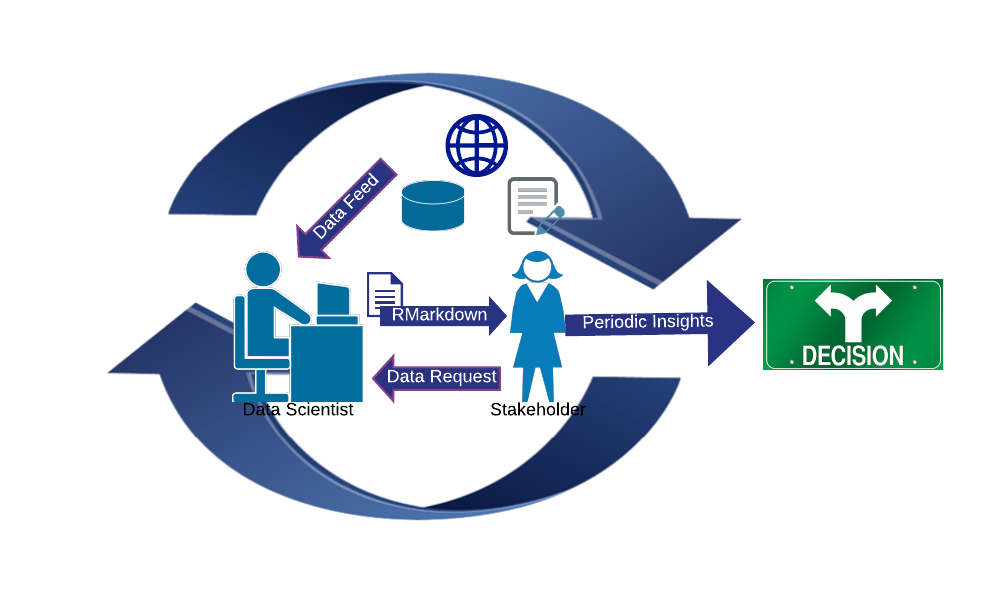


Figure 4. The Non-Operationalized Data Science Process

The issue with this approach, is that upon the event data changes, or your business requires these computations frequently, an ad-hoc computation by the data scientist is not sufficient to keep up with demand.

To keep up with today’s fast-paced environment, data scientists often work in-tandem with application development teams to operationalize data science work to production systems. The data scientist would build his/her algorithm, then pass it on to the development team to implement into a production system. This application would then be accessible by the business stakeholder for real-time insights and decision making (See Figure 5).

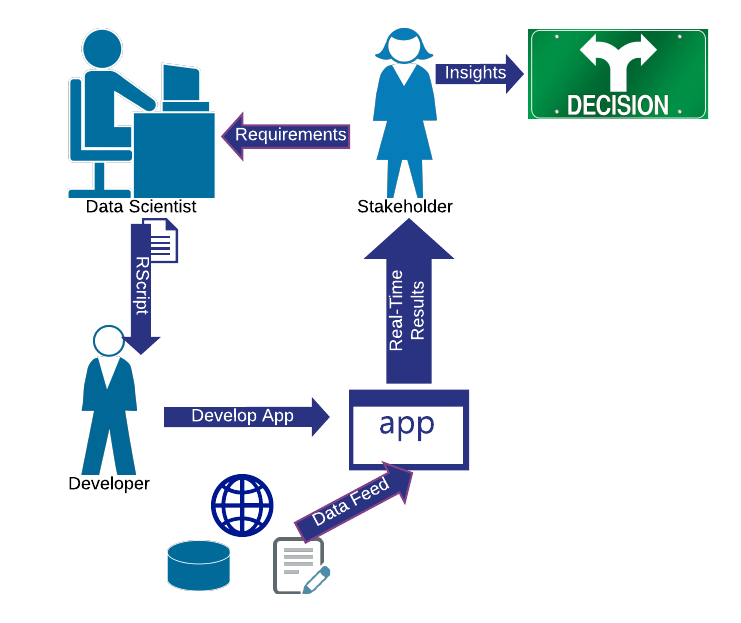


Figure 5. The Operationalized Data Science Process

This way of doing things allows the data scientist to focus on *providing new value* to the organization, rather than focusing on *maintaining* existing work they have done in the past. As seen in Figure 5, the continuous loop of requesting information from the data scientist is removed from the process and he/she was only involved in the initial RScript creation.

Operationalizing R Code is not, however, a new concept. Organizations have been implementing R code through frameworks such as shinyapps, R.Net packages, command line batch scripts, and various other methods. Many of these methods hinder the capabilities of a system implementation, and others, such as R.Net packages for the C# programming language, require a near re-write of the original RScript provided from the data scientist due to language syntax differences. These problems compound with limitations of the R language, such as data movement performance of large volume datasets since computations are not done in-database.

With R Services in SQL Server 2016, the Applications are capable of pushing all computations to the DBMS system. The Application may call a system-stored procedure which executes in the DBMS. This procedure may contain SQL transformations, data manipulation statements, and R Script code all together. The R Script will execute in-database, removing the aforementioned data movement issues (See Figure 6).

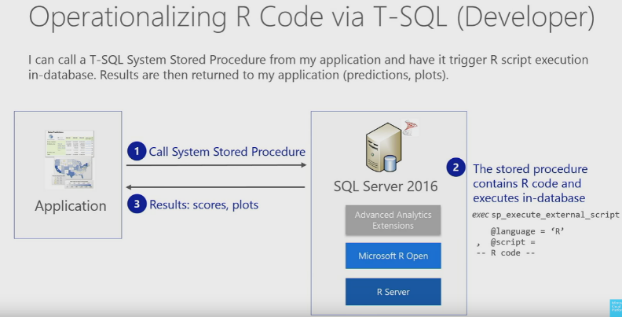


Figure 6. Operationalizing R Code Example  
(Courtesy of Elharrar, Microsoft Cloud Platform)

Result output of the R Script may include relational data values, R Model objects, Plot graphics, and ultimately any type of data that may be stored in an R variable. Another advantage seen with executing R scripts within SQL Server is the ease of transitioning the script from the R IDE to SQL Server. Most scripts are capable of being copy/pasted into the *@script* variable assignment without further modification. This makes maintaining the R code once in the production environment easy, as adjustments to the original script may be directly incorporated.

During this research, our team decided to put R Services into action with real-time Tumor Diagnosis predictions. All source code and reproducibility steps are outlined in our Project GitHub Repository ([MSDS7330\_RServicesDemo](https://github.com/amfrye777/MSDS7330_RServicesDemo)). See a high level overview of the steps taken to build this system below.

1. Create database with tables designed to store Tumor Statistics, R Plot Images, R Model Objects, and other miscellaneous dimensions necessary for a normalized design.[[7]](#endnote-7)
2. Create an R Discriminant Analysis Model Script to predict Tumor Diagnosis as "Benign" or "Malignant".
3. Create procedure to facilitate execution of R Script. R Script output contains a serialized value containing the R Model computed and BoxPlots for all explanatory variables. Procedure saves serialized R objects into the DBMS environment.[[8]](#endnote-8)
4. Create procedure to facilitate retreival of R Discriminant Analysis Model from DBMS table, passing input values needed for a prediction. Procedure output provides prediction and probability percentage computed.
5. Create report to execute prediction procedure(from step 4), passing user submitted values as inputs. Report displays prediction results, as well as plots for all explanatory variables passed in for additional context.[[9]](#endnote-9)

This demonstration was insightful, and blended a myriad of concepts discussed in this paper. In addition to enhancing performance of computations through in-database processing, reducing data movement latency, and operationalizing an R prediction model, we found an unexpected gain from this approach as well. We were able to *eliminate the* *overhead* of “Building” the model every time a prediction was needed to be made. The R Model object from the source tumor statistics data was saved in a table, and simply retrieved for the prediction procedure. In traditional R code in a workstation IDE and most other operationalizing frameworks, the model would require building every time a new R session is created. As data volume required for computations increases, this overhead reduction could attest to a huge performance buyback, compared to other frameworks.

# CONCLUSION

The ability to use the computational efficiency of R with the processing capacity of a Database Management System is remarkable. Our timed test of loading a data set of ten million records showed a significant reduction of wait time, a decrease of 94.1%. Then, our discriminant analysis on a real-world data set of tumor data demonstrated the ability to retrieve previously built models, versus re-building them every time a user opens an instance of R. The results from these tests show the impact this integration will have on combining R with SQL Server 2016. Regarding the areas of impact, one can deduce rather quickly the impact of this integration on data science. We also see the benefits to:

* Database developers, who are tasked with bringing together multiple technologies to deliver sharable results within an organization. This integration supports a developer's efforts to architect a way to support the data analysis efforts of others.
* Database administrators, who are concerned with data security, reliability, and allocation of resources. Since data is not loaded into a user's main memory, R sessions are run independently of the database processes, and the allocation of resources to R runtime can be set to keep database performance from suffering, a database administrator would certainly benefit from this integration.
* Data engineers, who must create a cohesive data platform with both competing and complementary technologies. The R Services integration enables this role to truly integrate certain systems versus having to only rely on parallel workflows within the same system.

While R is not a one-size-fits-all solution, the principle here is that integrations between data analysis tools and databases translate into new ways for a data scientist to bring value to their corporations and communities. Certainly, other integrations, present and future, will continue to be of utmost importance to promote integrity and reliability to the world of data science.

# References

1. D. Elharrar and U. Jayachandran, “R Services in SQL Server 2016,” *https://www.youtube.com/watch?v=8Sly49zDZEw*, Mar. 2016.
2. J Wilcox, “Pass Summit 2015 – Microsoft Foundation Session – Advanced Analytics,” *https://www.youtube.com/watch?v=8IW9FXGjxZY*, Time: 16:19 – 21:40, Oct. 2015.
3. P. Zhao. (2016, Sept). “R with Parallel Computing from User Perspectives,” *http://www.parallelr.com/r-with-parallel-computing/?\_utm\_source=1-2-2,* Sep. 2016.
4. T. Kastrun. (2016, Sept). “Introduction to Microsoft R Services in SQL Server 2016,” *http://www.sqlservercentral.com/articles/Microsoft/145393/,* Sep. 2016.
5. S. Weller, “A Simple Big Data Analysis Using the RevoScaleR package in Revolution R,” *https://www.r-bloggers.com/a-simple-big-data-analysis-using-the-revoscaler-package-in-revolution-r/,* May 2011.
6. R. Kabacoff. (2014, n.d.). Access to Database Management Systems (DBMS): ODBC Interface,” *http://www.statmethods.net/input/dbinterface.html,* 2014.

Data and Coding Citations:

Tumor Diagnosis Data – Operationalizing R

1. W. Woldberg, Dr., W. Nick Street, Olvi L. Mangasarian, “Wisconsin Prognostic Breast Cancer”, December 1995

Fraud Risk Data  - Data Movement Performance

1. Microsoft Corporation, “RevoScaleR SQL Server Getting Started Guide,”, *https://packages.revolutionanalytics.com/doc/8.0.0/win/RevoScaleR\_SqlServer\_Getting\_Started.pdf* , 2016

Saving serialized model coding help

1. Blue Granite, “Tutorial: SQL Server R Services,” *https://www.blue-granite.com/tutorials/sql-server-r-services*

Saving serialized plot coding help

1. K. Verbeek, “SQL Server 2016 R Services : Display R Plots in Reporting Services,” *https://www.mssqltips.com/sqlservertip/4127/sql-server-2016-r-services-display-r-plots-in-reporting-services/* , Feb. 2016.

R.Net C# Package

1. CodePlex.com, “R.Net,” https://rdotnet.codeplex.com/, Aug. 2015.

TumorDiagnosis GitHub Coding Repo

1. A. Frye, *https://github.com/amfrye777/MSDS7330\_RServicesDemo,* Oct. 2016.

Data Movement Citations:

Real-time replication, serial loading, or incremental transfer

1. R. Hodges and MC Brown, “Real Time Data Movement: The Key to Enabling Live Analytics with Hadoop,” *http://www.dbta.com/Editorial/Trends-and-Applications/Real-Time-Data-Movement-The-Key-to-Enabling-Live-Analytics-With-Hadoop-97095.aspx,* May 2014.

ETL and ELT

1. DBMS2.com, “Kinds of Data Integreation and Movement,” *http://www.dbms2.com/2012/03/12/kinds-of-data-integration-and-movement/* , Mar. 2012.
2. Microsoft Corporation, “Data Movement Patterns,” *https://msdn.microsoft.com/en-us/library/ff650072.aspx,* Jun. 2003.

RevoScaleR functionality

1. Microsoft Corporation, “RevoScaleR SQL Server Getting Started Guide,”, *https://packages.revolutionanalytics.com/doc/8.0.0/win/RevoScaleR\_SqlServer\_Getting\_Started.pdf* , 2016

Revolution R Services

1. Microsoft Corporation, “Revolution R Enterprise 7.5.0 Server (RRE-7.5.0)”, *https://www.microsoft.com/en-us/download/details.aspx?id=49505*
2. Microsoft Corporation, “Revolution R Enterprise 8.0 Server (RRE-8.0.1)”, *https://www.microsoft.com/en-us/download/details.aspx?id=51204*

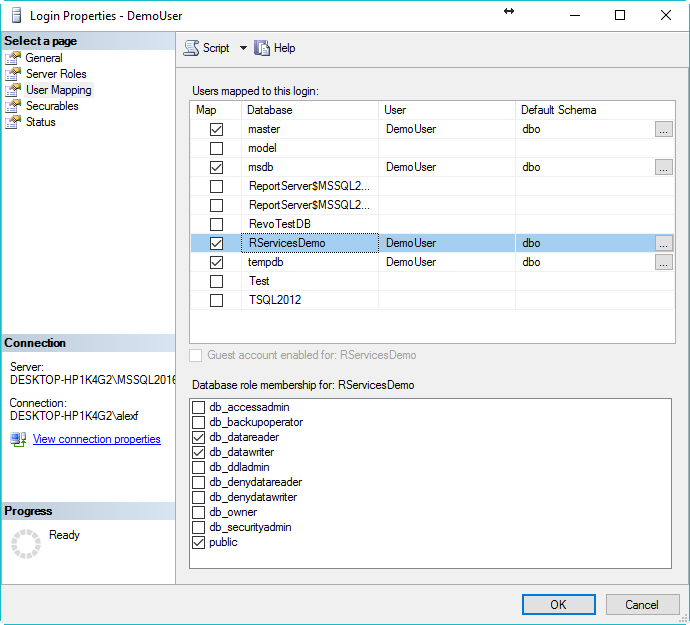
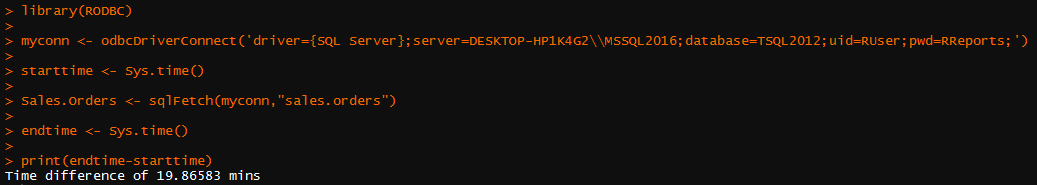
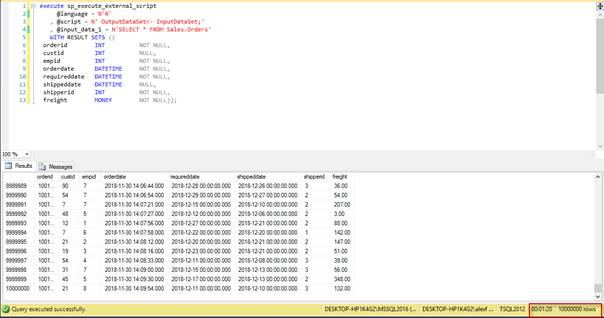
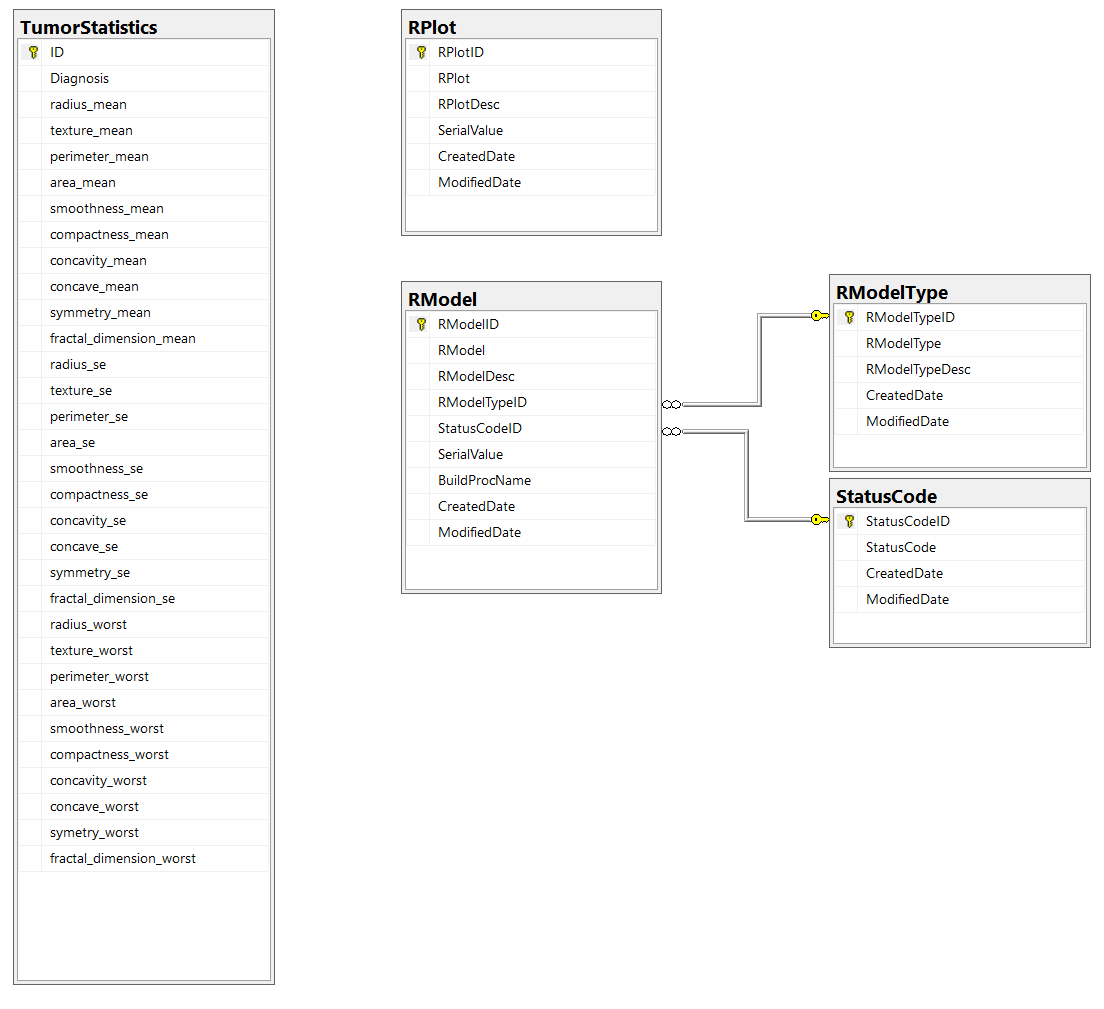
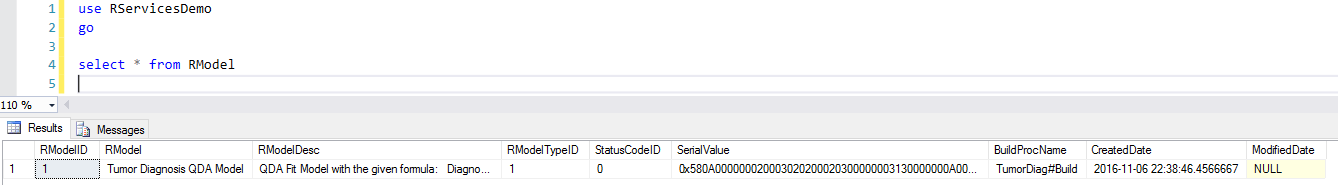
Scale / Performance Citations

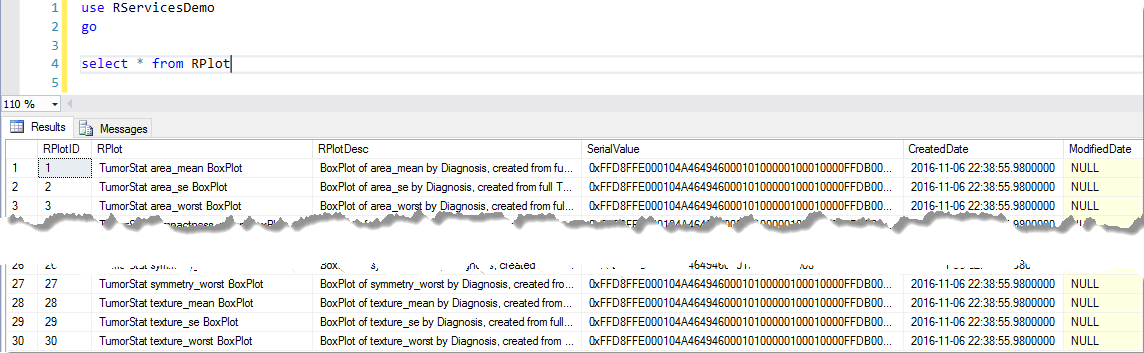
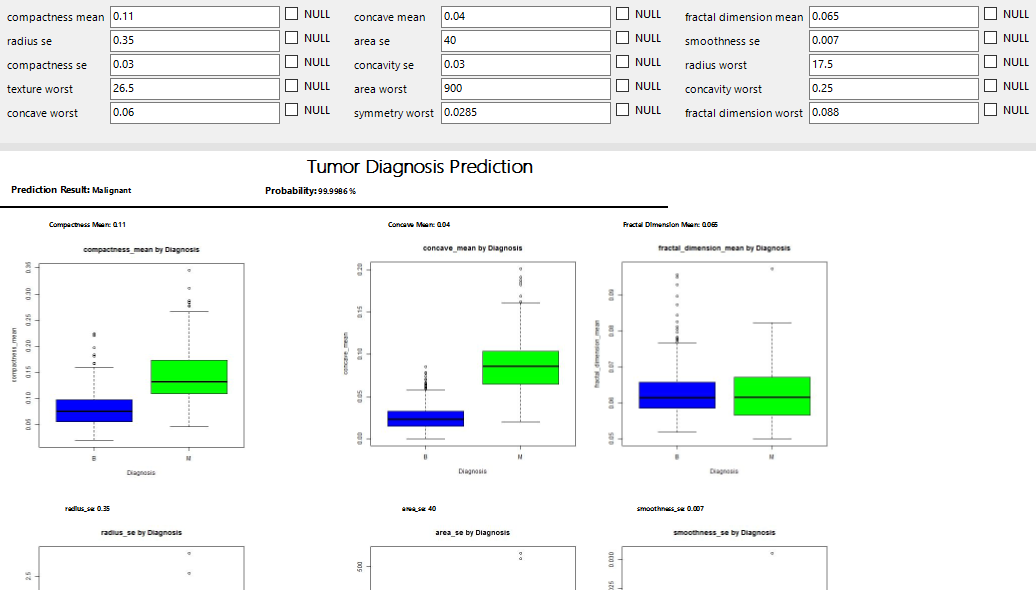
1. R Documentation, “Memory Limits in R,” *https://stat.ethz.ch/R-manual/R-devel/library/base/html/Memory-limits.html*
2. P. White, “Understanding and Using Parallelism in SQL Server,” https://www.simple-talk.com/sql/learn-sql-server/understanding-and-using-parallelism-in-sql-server/, Mar. 2011.
3. Microsoft Corporation, “R and Data Optimization (R Services),” https://msdn.microsoft.com/en-us/library/mt723575.aspx, Nov. 2016

Background:

1. Microsoft Corporation, “Data Science Deep Dive: Using the RevoScaleR Packages,” *https://msdn.microsoft.com/en-us/library/mt637368.aspx*, Sep. 2016

# EndNote References

1. [↑](#footnote-ref-2)
2. D. Merrill, Ph.D, “R is Not Enough for Big Data.” *http://www.forbes.com/sites/douglasmerrill/2012/05/01/r-is-not-enough-for-big-data/#2c169a61725a* , May 2012.  
    [↑](#endnote-ref-2)
3. P. Zhao. (2016, Sept). “R with Parallel Computing from User Perspectives,” *http://www.parallelr.com/r-with-parallel-computing/?\_utm\_source=1-2-2,* Sep. 2016. [↑](#endnote-ref-3)
4. User Privileges in SQL Server  
    [↑](#endnote-ref-4)
5. Data Movement Test of 10 Million Records to R Studio IDE (19 Min 52 Sec)  
    [↑](#endnote-ref-5)
6. Data Movement Test of 10 Million Records Utilizing R Services In-Database (1 Min 20 Sec)  
    [↑](#endnote-ref-6)
7. RServicesDemo Database Table Design  
    [↑](#endnote-ref-7)
8. DBMS Table Sample Data  
   

    [↑](#endnote-ref-8)
9. Report Output Example  
    [↑](#endnote-ref-9)