

R

Contents

Introduction

Setting up a data science virtual machine

Accessing SQL Server data from R

Embedding R in SQL Server functions and procedures

R in Power BI

Installing R with SQL Server 2016 RC3

Introduction

This guide combines the worlds of SQL Server and R to enable us to exploit the power of both and to get a better understanding of the data that drives our organisations. SQL specifically T-SQL is great for certain kinds of operations but was never designed to do the kinds mathematics and statistics and charting that are available in R.

Traditionally R programmers have connected to databases via odbc and pulled the data they need into R using the rodbc library. However SQL Server can be configured to call R from within stored procedures and functions which allows us to put the output form these back into SQL as part of process.

Microsoft has embedded R into SQL Server by leveraging R Enterprise[[1]](#footnote-2) and implementation of R that uses disk as well as RAM to enable analysis of very large volumes of data.

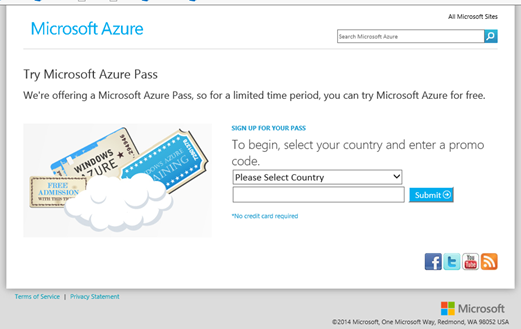
You have two options to complete these labs either use the [R tools in Visual Studio 2015](https://www.visualstudio.com/en-us/features/rtvs-vs.aspx) or [R Studio](https://www.rstudio.com/). There is no right option but if you are familiar with development in Visual Studio. This guide is based on the data science VM in Azure which is already partially setup for Visual Studio 2015 but it’s an easy matter to add R Studio and there are steps for this.

Typically, when we do data projects we remotely manage the servers and services we use and in this lab there is already a SQL Server 2016 server complete with R integration already configured, with various sample databases and scripts. There are some notes at the back of this guide to help setup your own if you wish after today so you can continue to work.

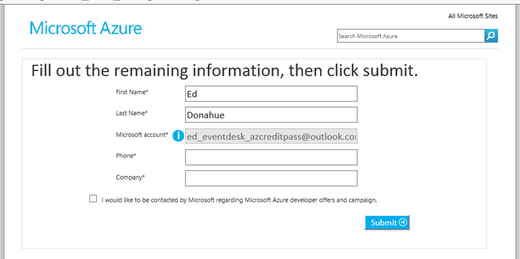
Setting up a Data Science VM in Azure

You may already have an Azure subscription for example with MSDN or as part of your business however it might be easier if you used the Azure Pass supplied as part of the workshop. This gives you a 1 month $100 usage cap (whichever is the sooner) and will isolate any work you do in the workshop form any resources you have in Azure already.

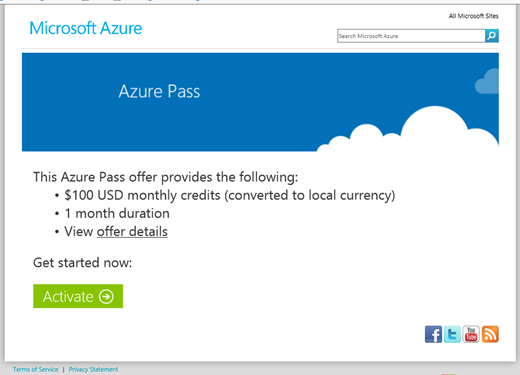
1. Open an **inPrivate** or incognito session in your browser and navigate to [www.microsoftazurepass.com](http://www.microsoftazurepass.com).



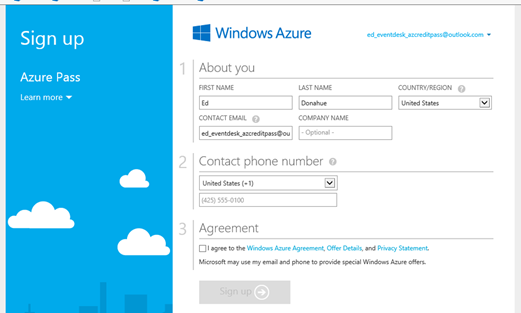
1. Select your country and enter the promo code you have been given and click **submit**.



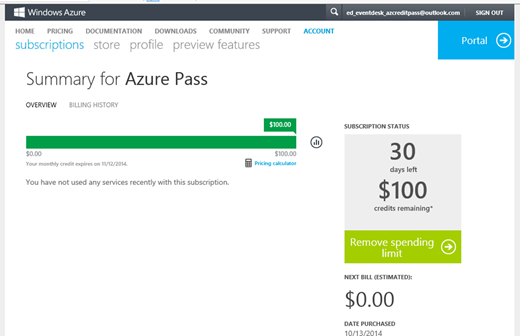
1. Sign in with a Microsoft account to continue. If you have an account password saved or you automatically sign into other Microsoft services that Microsoft account will be used to redeem the promo code.
2. Fill out the remaining information and click submit.



1. Review the Azure Pass offer and click **Activate**.

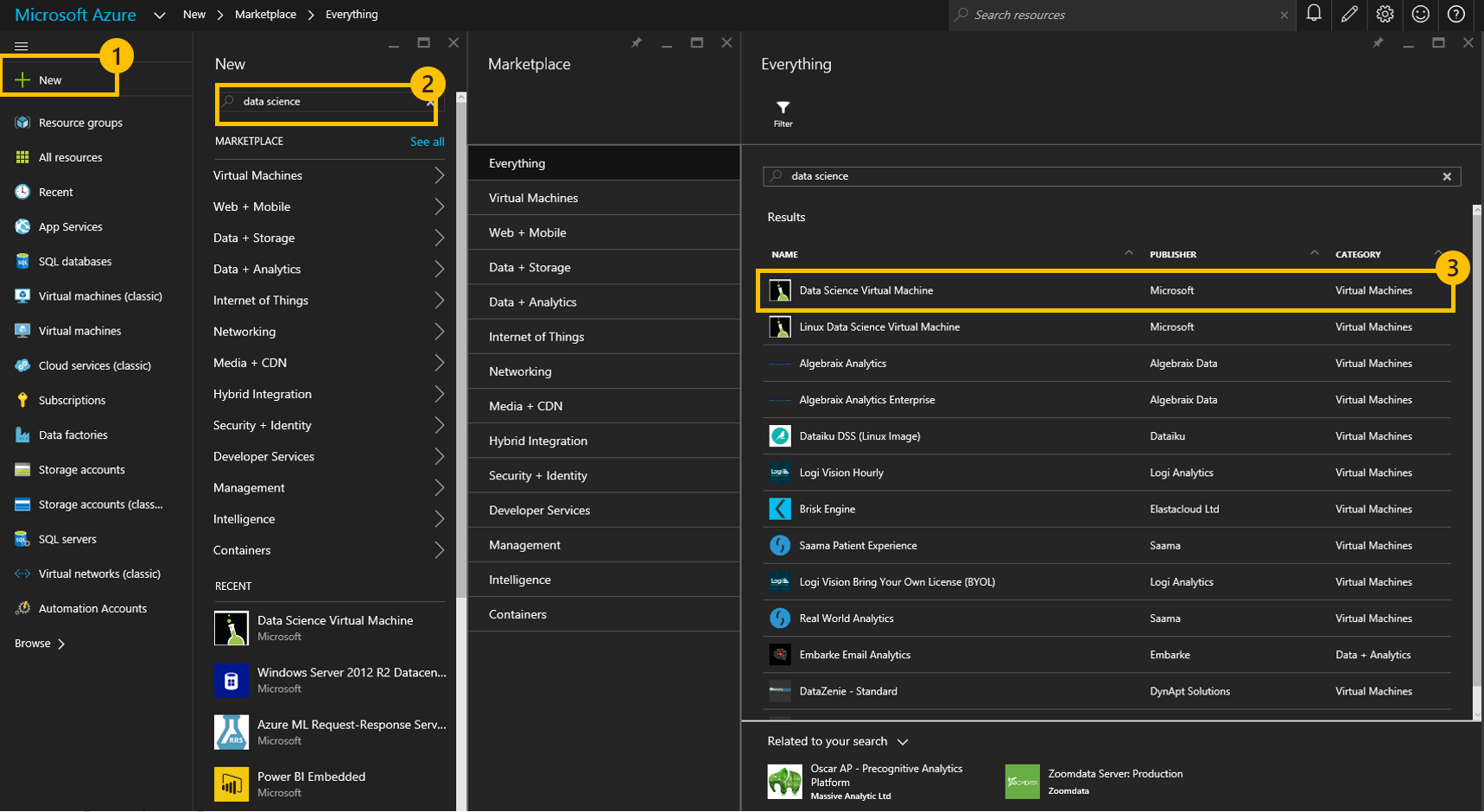


1. Fill out the rest of the required information and click **Sign up.**
2. The subscription will take 3-10 minutes to activate and you should then be redirected to a summary page like this..



After a few minutes your trial Azure subscription will be created and the first thing we’ll want to do is to create a setup where we can work with R and the various Microsoft solutions that have integration with R. The easiest way to do this is to use a Virtual Machine as this will avoid us having to install anything locally – we’ll just use a remote desktop connection to this new VM. Also there are two dedicated data science VMs in the Azure Gallery one based on Windows and one on Linux. Today we’ll be using the Windows one.

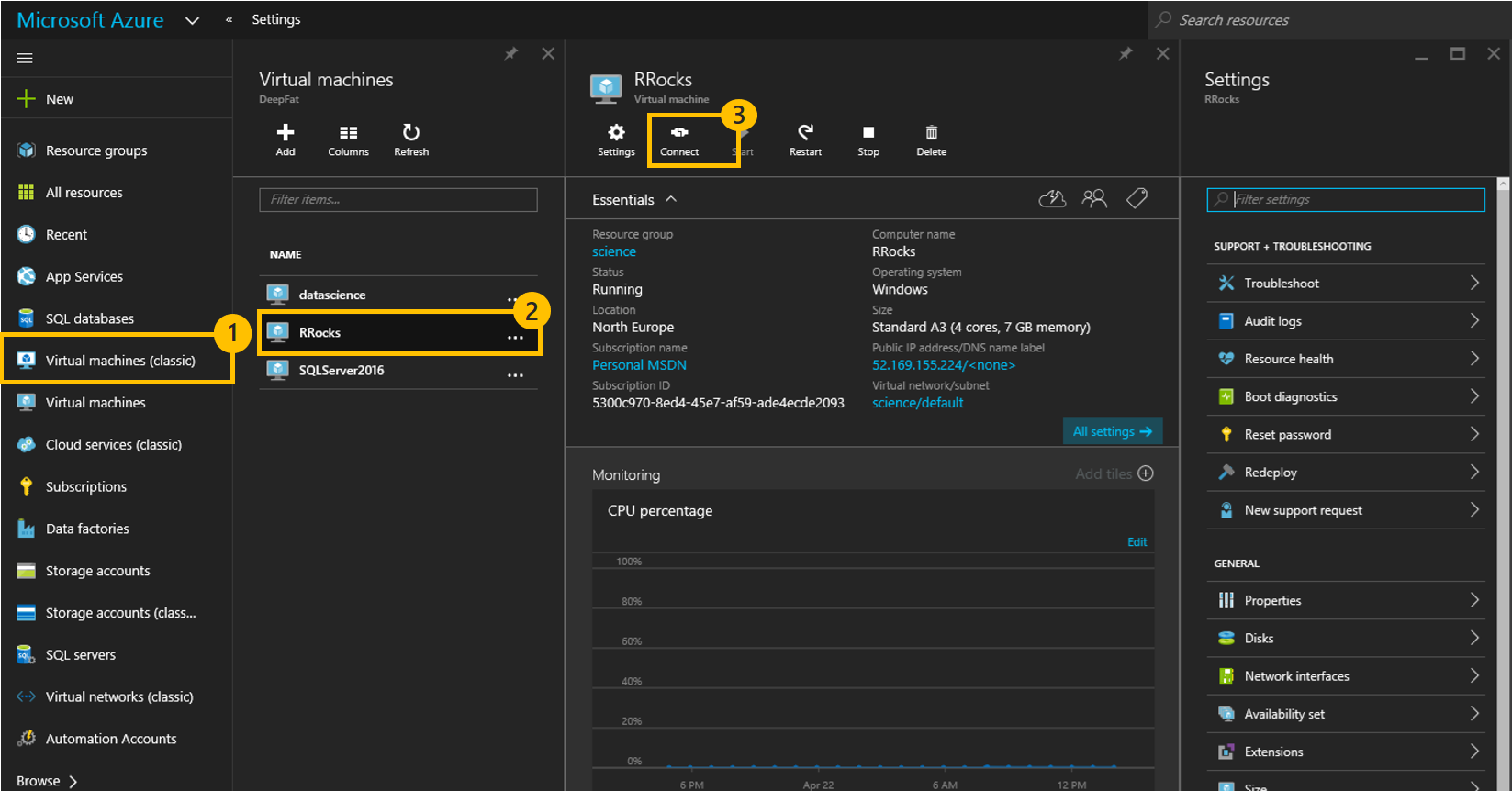
The Azure subscription activation process should redirect us to the portal but if not go to [www.portal.azure.com](http://www.portal.azure.com). From there we can simply select New, and search for “data science” to find the VM we want..



Review the information about the VM and then click create to open the blade (the fly out tabs in the Azure portal) to specify the configuration of the VM..

|  |  |
| --- | --- |
|  | Give the VM a meaningful name, user and strong password.  The resource group is just a container with specified assets as this is a new subscription we’ll need to create one of those give it a meaningful name like science.  The location is where the VM will live so Northern Europe as that is where the other assets are that we’ll be using today.  Click OK |
|  | Select A3 , and after today remember to turn this off until you need it again so you get the maximum benefit of the trial.  Click OK to confirm |
|  | Normally we’d change these for a production environment but we can leave them as is for this lab.  Click OK to confirm. |
|  | Review the Summary information.  click OK |

Review the Purchase Information and click Purchase to start deploying the VM. This will take 5 minutes or so and you should then be able to expand Virtual Machines in the portal and see your new VM in this case RRocks. We can stop it start it from here but for now we want to connect and when we click on the connect icon we’ll download an rdp connection for the VM (into downloaded files on Windows PCs)



Locate the <name of your VM>.rdp file and enter the login credential used to create it (use .\login name if you are on a domain joined laptop).

When you connect you’ll see that a variety of useful shortcuts on the desktop, and there are more on the start menu. This VM is based on Windows Server 2012R2 and the first thing we need to do for this lab is disable enhanced security as we’ll be going to various websites and downloading examples and utilities. Go to Server Manager and on the local server tab click on IE enhanced Security and disable it for administrator.

**Note** this VM is being constantly updated so it will eventually have SQL Server 2016 tooling on it complete with R integration already installed and Microsoft R Open server.

R includes a mark-up language, RMarkdown, to enable output to be rendered as web pages and if we want this to work form inside Visual Studio then we’ll need an open source utility [pandocs](https://github.com/jgm/pandoc/releases/tag/1.17.0.2). This utility is included in R Studio so probably the easiest thing might be to install R Studio on the VM and then make a choice about which to use.

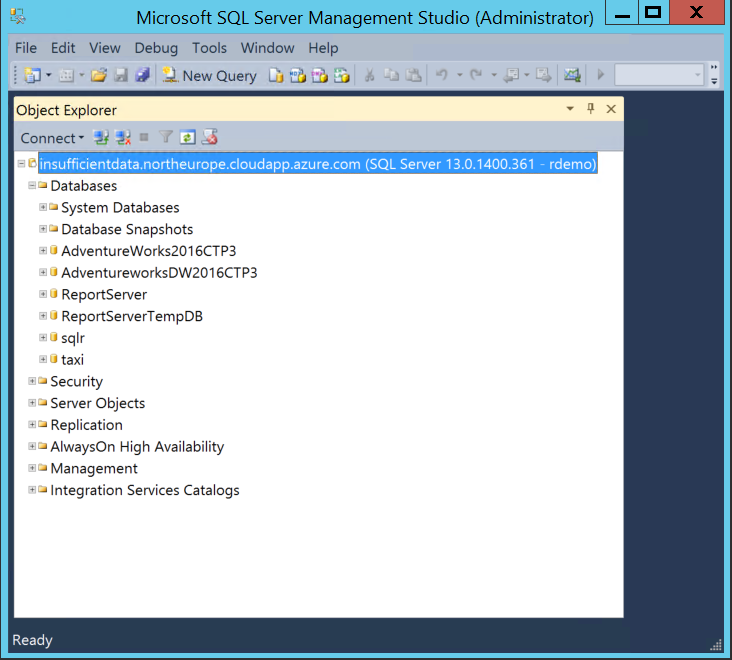
Connect to the data science VM and go to [www.rstudio.com](http://www.rstudio.com) and download RStudio desktop Open Source Edition.

Visual Studio 2015 Community Edition is already installed complete with R & Python tools but we’ll need to sign in with a Microsoft account to use it, for example the one we used to create the Azure Subscription we are using.

Using SQL Server data in R

The traditional way to use data in SQL database with R is to do this via an ODBC connection. R has an RODBC library for this and in this short exercise we’ll see how to use that. One of the features we have in R Enterprise is to specify where the R is executed (on the client or on a remote server) and a set of common Rx functions to replace the equivalent R function for improved performance and scale. A good example of that is that in R we would access data in SQL Server with the rodbc library where in R Enterprise there are is the RxSQLServerData to define the SQL server data source and RxinSQLServer to specify where the R should be executed locally or on a remote server. There are also Rx functions to specifically connect to Hadoop & Terradata.

First we should confirm we can connect to the sample database. Open SQL Server Management Studio and click on connect. Use SQL Server Authentication and login as rdemo / SQLbits16!..



Now we know we can connect with these credentials we can see how to do it in R and make a connection to the sqlr database.

sqlConnString <- "Driver=SQL Server; Server=insufficientdata.northeurope.cloudapp.azure.com; Database=sqlr;Uid=rdemo;Pwd=SQLbits16!"

Now we know we can declare an RxSQLServerData object using a table, view or query in a given database on that server in this case to connect to the Fraud table in that database..

tblFraud <- RxSqlServerData(connectionString = sqlConnString, table = "dbo.Fraud", rowsPerRead=5000)

ow Now we can interrogate how R understands this data with another Rx command RxGetVarInfo..

rxGetVarInfo(data = tblFraud)

Var 1: VendorNumber, Type: character

Var 2: VoucherNumber, Type: integer

Var 3: CheckNumber, Type: integer

Var 4: InvoiceNumber, Type: integer

Var 5: InvoiceDate, Type: character

Var 6: PaymentDate, Type: character

Var 7: DueDate, Type: character

Var 8: InvoiceAmount, Type: numeric

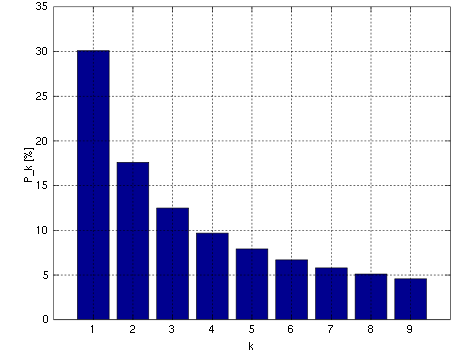
Var 9: PONumber, Type: integer

If we want to use this table as a data frame in R then we need to import with the RxImport function :

dfFraud <- rxImport(tblFraud)

Note this loads the data in the chunk size we specified and now we can use this data just as we would any other data frame in R.

For example we can plot the data to look at the distribution of invoice amounts for a given vendor and compare that to a special distribution used in fraud the Benford distribution which states that the leading digit of an amount (in our case the amount of the invoice will not be random but will look like this



To do this we will create a new object tblVendor score and this time specify a query we can get the data for just one vendor (Vendor ID 105436) where there is already a function to get the leading digit of the invoice amount[[2]](#footnote-3)

sqlSingleVendor =

"select Top 100 Freq from VendorInvoiceDigits('105436') order by Digits" tblSingleVendor =

RxSqlServerData(connectionString = sqlConnString, sqlQuery = sqlSingleVendor)

dfSingleVendor <- rxImport(tblSingleVendor)

We can then use the plotting functions in R to show us how the two distributions compare visually. We’ll be using the ggplot2, ggthemes and reshape libraries so we’ll need to add these to our R workspace..

install.packages("(ggplot2")

install.packages("ggthemes")

install.packages("reshape2")

require(ggplot2)

require(ggthemes)

require(reshape2)

Now we can use this script to do the plots..

qq = as.numeric(dfSingleVendor$Freq)

pp = data.frame(num = factor(1:9), pct = round(100 \* (log(1 + 1 / (1:9)) / log(10))))

pp = data.frame(num = pp$num, Actual = round(100 \* qq / sum(qq)), Expected = pp$pct)

pp = melt(pp)

title = "Distribution of Leading Digits in Invoices"

gg = ggplot(pp, aes(x = num, y = value, fill = variable)) + geom\_bar(stat = "identity",

position = "dodge", alpha = 0.85)

gg = gg + labs(x = "Leading Digit", y = "Percent")

windowsFonts(Verdana = "TT Verdana")

gg = gg + theme\_igray(base\_size = 16, base\_family = "Verdana")

gg = gg + theme(legend.title = element\_blank())

gg

So what’s going on here? The easiest way to uncover this is to run each line and interrogate the output. The first time the variable pp is declared we get the Benfield distribution of numbers 1-9 based on a log curve just as per the histogram earlier.

num pct

1 1 30

2 2 18

3 3 12

4 4 10

5 5 8

6 6 7

7 7 6

8 8 5

9 9 5

Pp is then set to a three column data frame:

* The number being analysed (1:9)
* a column for the actual distribution (round(100 \* qq / sum(qq)))
* an Expected column from the distribution in the line above.

num Actual Expected

1 1 1 30

2 2 96 18

3 3 0 12

4 4 0 10

5 5 0 8

6 6 0 7

7 7 1 6

8 8 1 5

9 9 0 5

The variable pp is then transformed into the shape ggplot2 needs in order to plot it with the melt function and now looks like this..

num variable variable value

1 1 Actual value 1

2 2 Actual value 96

3 3 Actual value 0

4 4 Actual value 0

5 5 Actual value 0

6 6 Actual value 0

7 7 Actual value 1

8 8 Actual value 1

9 9 Actual value 0

10 1 Expected value 30

11 2 Expected value 18

12 3 Expected value 12

13 4 Expected value 10

14 5 Expected value 8

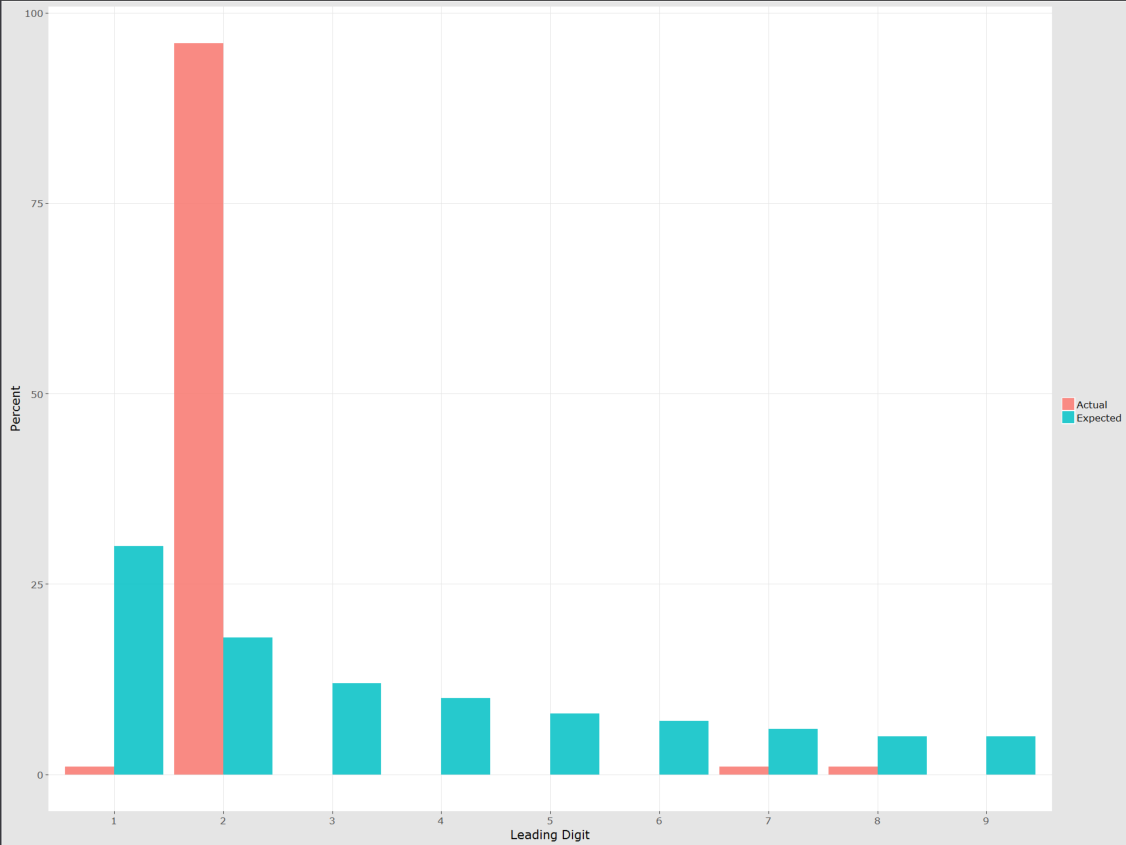
15 6 Expected value 7

16 7 Expected value 6

17 8 Expected value 5

18 9 Expected value 5

The gg variable contains our plot so it uses pp where num is the number (0:9) we are analysing and we are plotting the value (percentages of times a number occurs) against the two variables (Actual and Expected). When run the last line to return gg we can see the plot in this case in Visual Studio..



So clearly there is something very odd with the invoices for this vendor and we should investigate further.

Using R in SQL Server 2016

What we did in the last section isn’t actually that much use to our users as they don’t want to be using Visual Studio to look at these plots. So what we need to do is to provide them access to this data in a familiar environment e.g. a browser. If we were doing this in house/on premises then this is all possible from within SQL Server 2016::

* Embedding the R code we have in SQL rather than embedding SQL in R as we did in the last section.
* Using Reporting Services to show the plots.

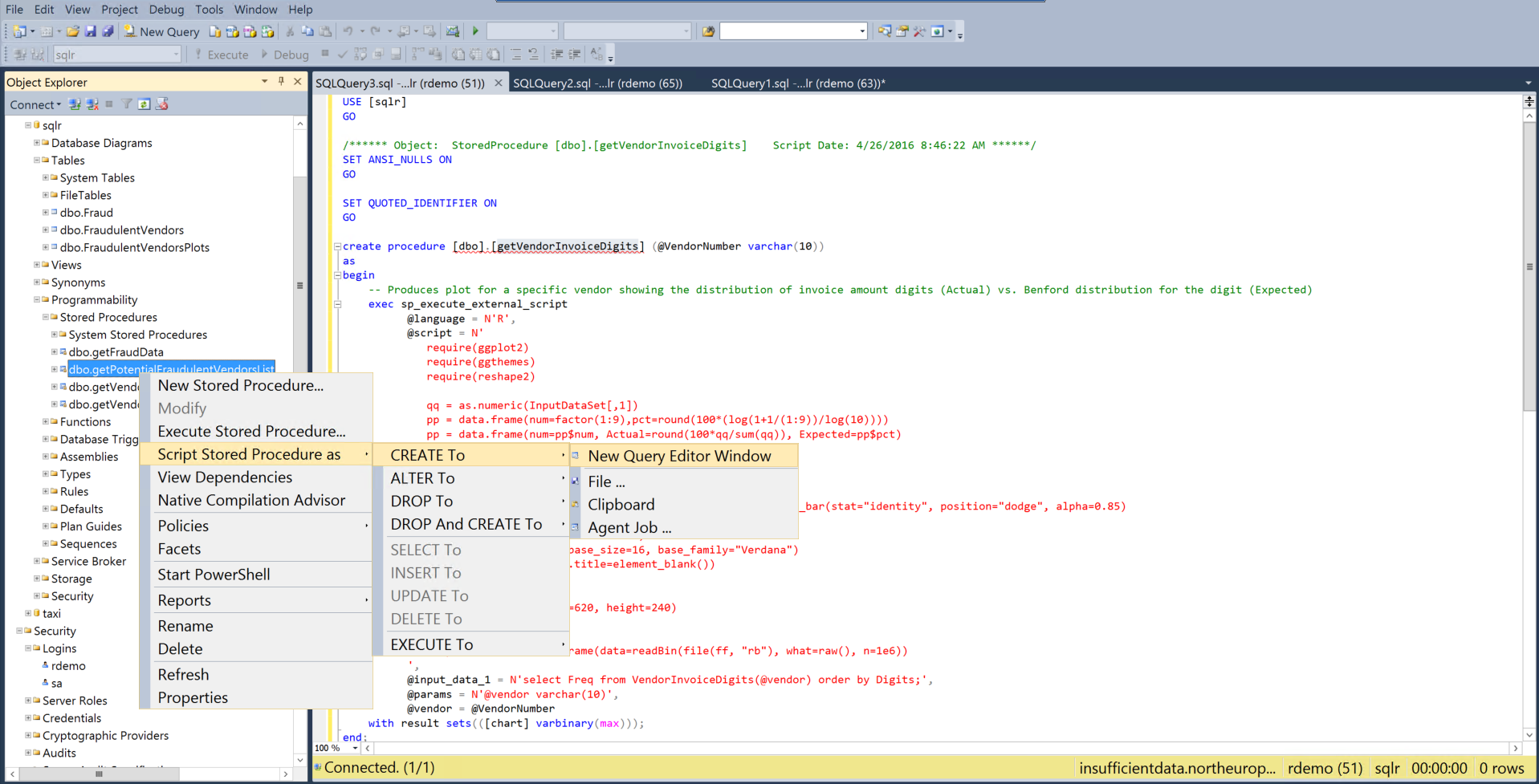
|  |
| --- |
| This lab uses a separate preconfigured SQL Server 2016RC3 VM and there are notes on how to do this and the end of this guide and these libraries are already setup. |

Before we start doing that remember that we made some changes to the data science VM in order to be able to run plots etc. by installing various library packages. If we now want to run our code on a remote server to provide these plots to our users that server must be similarly configured, so bear this in mind when implementing R in production.

To call the code we used in the last section we would embed it in a stored procedure so let’s see how that can be done by opening SQL Server Management Studio and connecting to the example SQL Server :

|  |  |
| --- | --- |
| **Server name:** | insufficientdata.northeurope.cloudapp.azure.com |
| **Login:** | rdemo |
| **Password:** | SQLbits16! |

Expand the rsql -> Programmability -> Stored Procedures and right click on getVendorInvoiceDigits. Right click on it and select Script Stored Procedure as -> as Create To -> New Query Editor Window..



As we can see embedding R in SQL is actually very much like embedding SQL in R in that we’ll have to build up the commands we want in a string and pass it into

exec sp\_execute\_external\_script :

@language = N'R',

@script = N'

require(ggplot2)

require(ggthemes)

require(reshape2)

qq = as.numeric(InputDataSet[,1])

pp = data.frame(num=factor(1:9),pct=round(100\*(log(1+1/(1:9))/log(10))))

pp = data.frame(num=pp$num, Actual=round(100\*qq/sum(qq)), Expected=pp$pct)

pp = melt(pp)

title = "Distribution of Leading Digits in Invoices"

gg = ggplot(pp, aes(x=num, y=value, fill=variable)) + geom\_bar(stat="identity", position="dodge",

alpha=0.85)

gg = gg + labs(x="Leading Digit", y="Percent") windowsFonts(Verdana="TT Verdana")

gg = gg + theme\_igray(base\_size=16, base\_family="Verdana")

gg = gg + theme(legend.title=element\_blank())

ff = tempfile()

png(filename=ff, width=620, height=240)

print(gg)

dev.off()

OutputDataSet = data.frame(data=readBin(file(ff, "rb"), what=raw(), n=1e6))

',

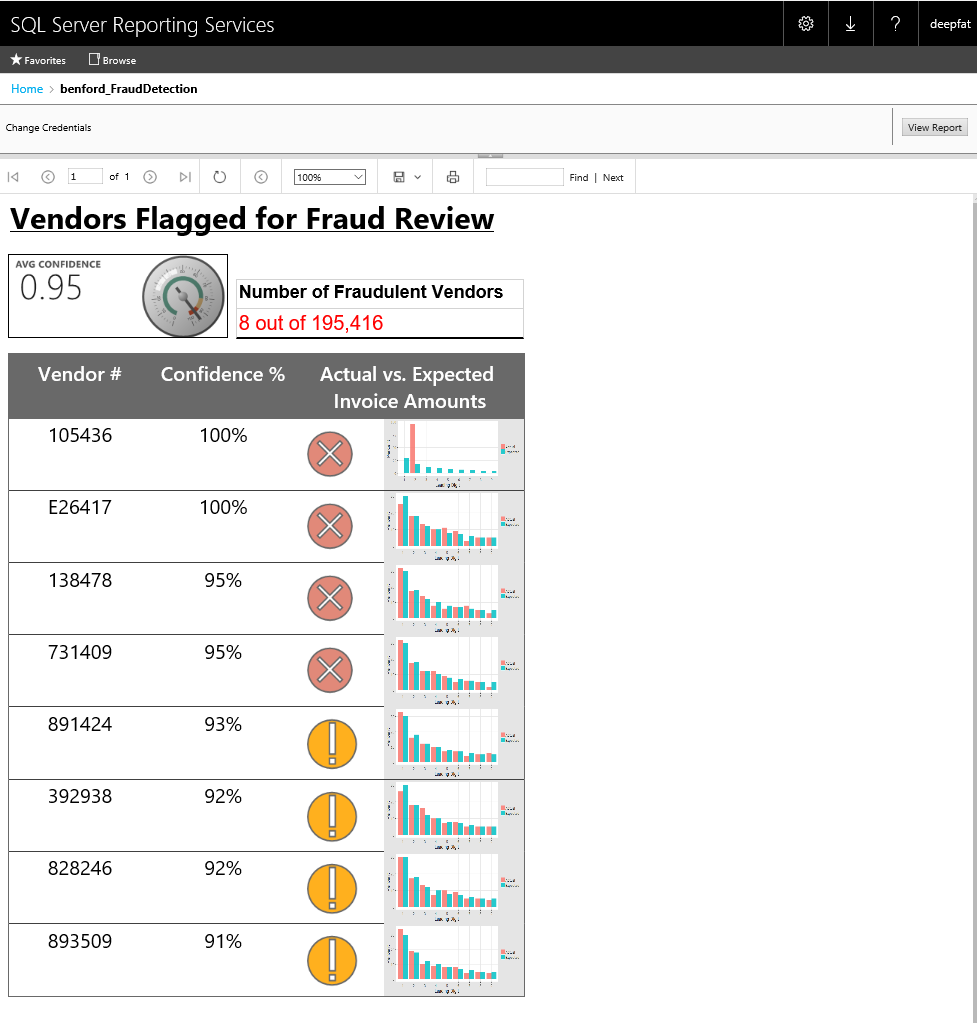
@input\_data\_1 = N'select Freq from VendorInvoiceDigits(@vendor) order by Digits;',

@params = N'@vendor varchar(10)',

@vendor = @VendorNumber

This code has additional lines at the end to convert the plot to .png format and return it as part of the data set. This stored procedure is then consumed in a loop for each vendor and the output ends up in a table, FraudulentVendorsPlots which has the vendor ID and a plot column in with the png format data in.

This plot data is then consumed in a reporting services report ..



Which simply renders the png files alongside each vendot

There is an excellent example of how to build a predictive model from the SQL Server team on Github and we can quickly pull this down onto our workspace VM with the following powershell:

$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

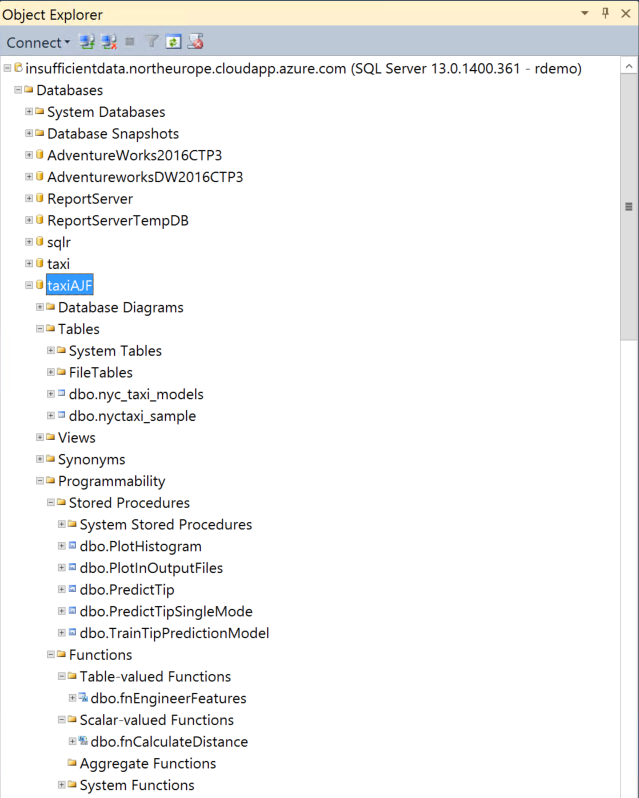
$wc.DownloadFile($source, $ps1\_dest)

.\Download\_Scripts\_SQL\_Walkthrough.ps1 –DestDir ‘C:\tempRSQL’

Where the demo files end up on our local drive (c:\tempRSQL). This demo is built around [taxis cab rides in New York](https://blogs.technet.microsoft.com/machinelearning/2015/04/02/building-azure-ml-models-on-the-nyc-taxi-dataset/) and includes all of the code and sample data we need and a PowerShell script (RunsSQL\_SQLWalkthrough.ps1) to deploy a sample database we can use. Run this script with the following parameters

|  |  |
| --- | --- |
| **Database server name:** | insufficientdata.northeurope.cloudapp.azure.com |
| **Database name:** | Taxi<Name> where <Name is yourName>? |
| **Login with prviliges to create databases:** | rdemo |
| **Password:** | Passw0rd ! |
| **Path to the csv file you want to upload** | C:\tempRSQL\nyctaxi1pct.csv |

If we open SQL Server Management Studio again we can see our new database (in this case TaxiAJF)..



We also have all the code to create this which we can pick apart. For now we can start to explore the data with a simple summary report to see how the fares vary by the aount of passengers that were carried:

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

However what really affects taxi fares is the trip distance not the number of passengers and there is a function to compute the “as the crow flies” distance for each trip fnCalculateDistance

which simply uses T-SQL trigonometric functions against the latitude and longitudes of the pickup and drop off points. If we want to do the same thing in R it wil be much faster and look like this:

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)

}

Why wold we do this in R instead of SQL, because it is much much faster and more accurate (the R is using great circles not simple trig), even if we used the geospatial functions.

The code to do this is not included here but if you have the basics down it should be possible to adapt this R function into a scalar T-SQL function which you may wish to attempt if you have time.

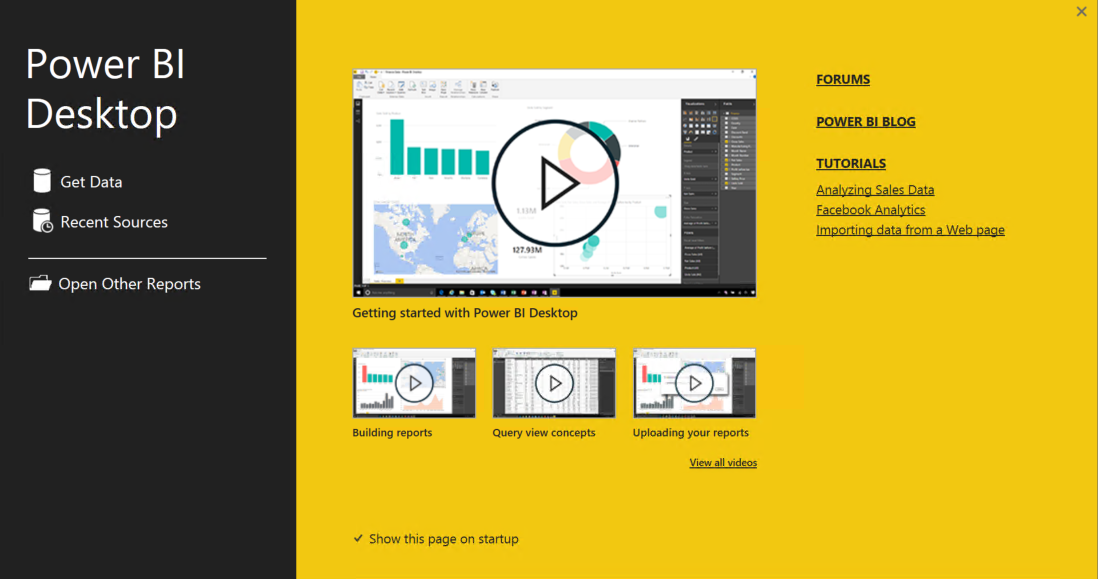
Using R in Power BI

Power BI is a powerful cloud based visualisation tool that is part of the Cortana Intelligence Suite. Visualisations can be designed form a single data source from any modern web browser and there is desktop designer that allows data to be manipulated and then presented before being uploaded to the service as a .pbix file. While there are rich charting types included in the tool and the ability to create new ones, using the Power BI SDK, many data analysts want to be able to use the charting capabilities in R. Another reason for using R with PowerBi is to use the maths & stats function in R to derive new insights which can then be plotted with R or the tools already in Power BI.

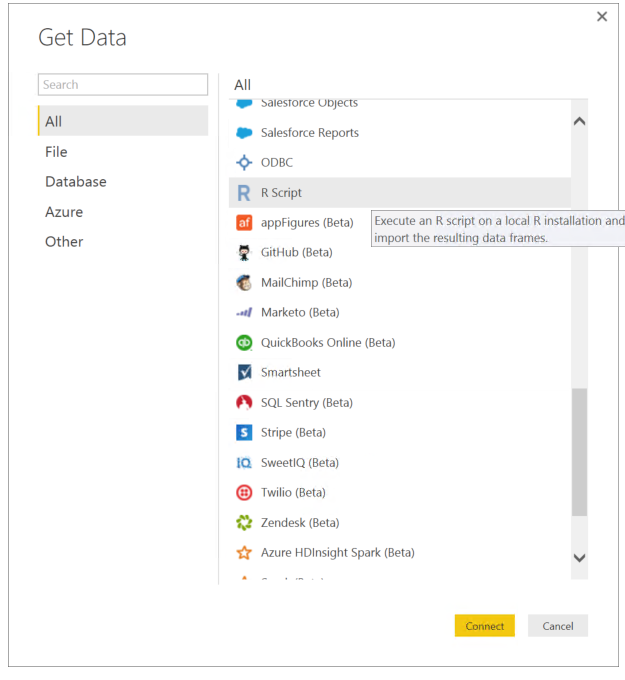
So in this section we’ll see how to embed those in Power BI and thus share our analytics with our users.

We’ll be using the Power BI Desktop and although this is pre-installed on the data science VM we have been using it will probably update when we first use it as there are new features being added all the time for example R integration! Open Power BI desktop and once the icon appears to update it close the Desktop and follow the links on the website to download it again and install it with defaults.

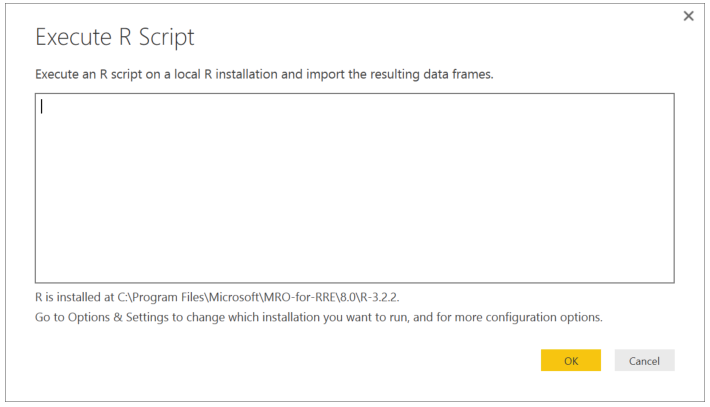
Open PowerBI desktop a second time and select Get Data frm the flash screen..



Notice we have a huge choice for using data in Power BI and one of those options is an R Script:



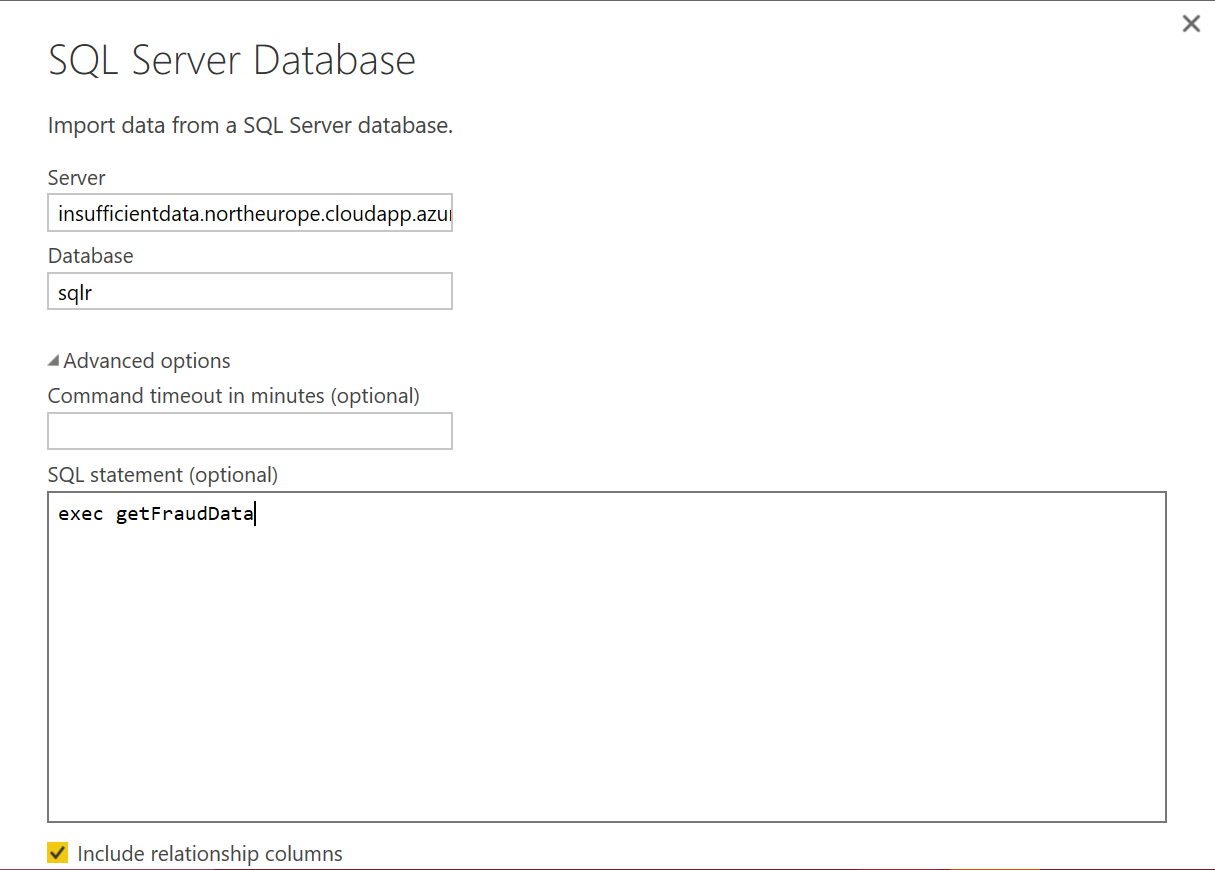
Select R Script and click Connect. We’ll get a simple window to paste our text into:



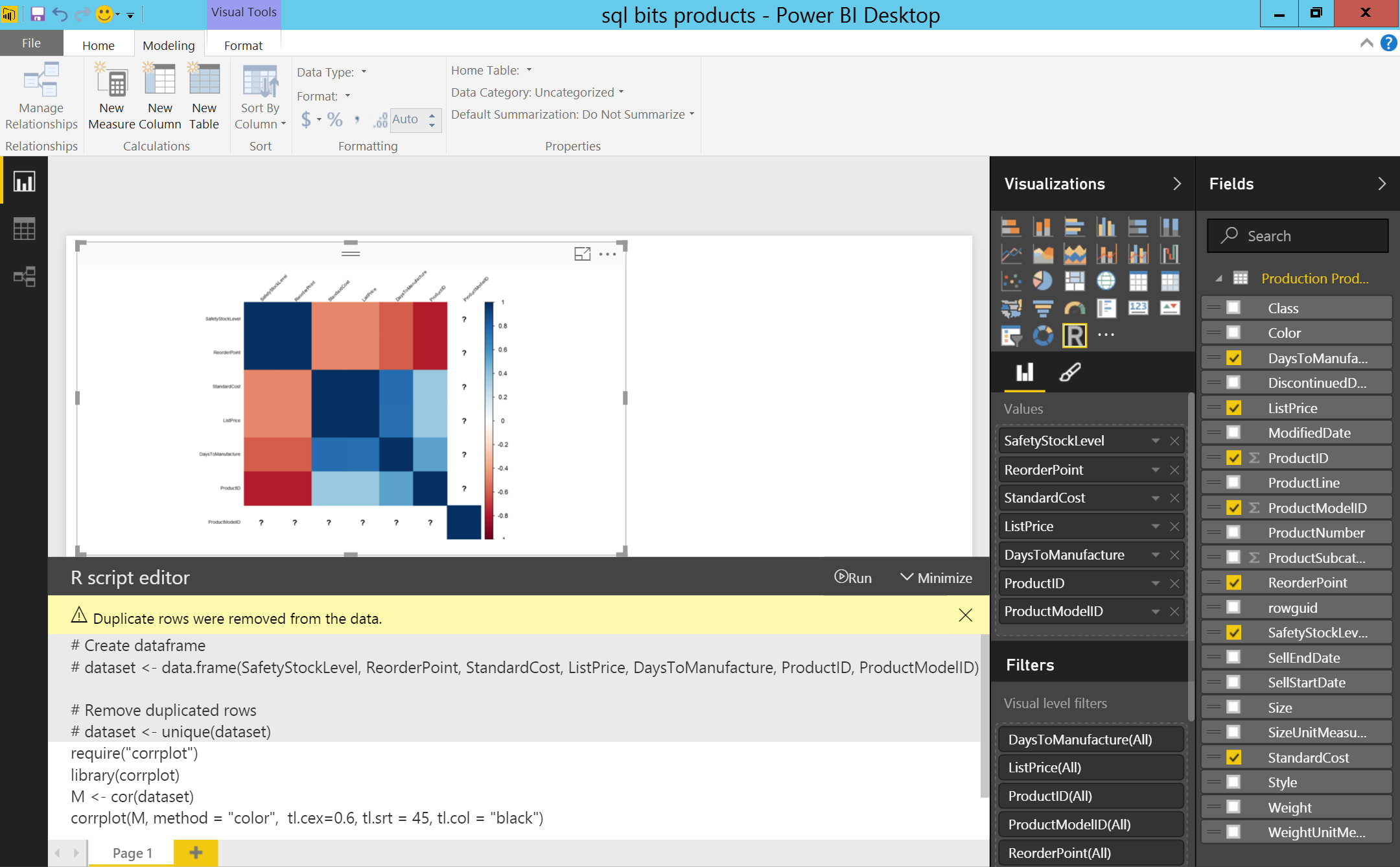
And a note telling us where R is installed and how we can change settings if we need to. Before we get started review these points to understand what we can do with R in Power BI and its limitations:

* The R is going to execute in the context we used when we uploaded .pbix file to the cloud service. So for a production service we need to have a way to get back to that and this is done with the Power BI gateway. This tool allows us to grab data from our network and put it into the cloud service.
* Any libraries we need must be explicitly referenced
* Only data frames can be used as data sources
* The full path to our R setup is needed.
* R scripts cannot be interactive which makes sense as there won’t be anyone to interact with them in the cloud service
* R scripts will time out after 30 minutes which also makes sense as no one would wait for a web site for anything like that amount of time.

We can now embed some of the code form earlier in the day. However if we are getting data form SQL Server a better way of using R is simply to call a stored procedure with R embedded in it so the code is running server side. To do that all we need to do is specify SLQ Server as the source and run a query to get the data.:



Another approach is to plot using R against data that is already in Power BI. There is an R visualisation tool in Power BI for this purpose and if you add one of these to a report and drag fields from data onto it these will form a data frame which can then be consumed by an R visualisation. In this example R has been used to look at the relationship between the numeric fields in the Adventure Works product table..



The fields we have selected appear in the grey area at the top of the code window which we cannot edit we do that by altering which fields are in the visualisation just as we would for any visualisation. Here just a few lines of R, in tis case using the corrplot library have been used ..

require("corrplot")

library(corrplot)

M <- cor(dataset)

corrplot(M, method = "color", tl.cex=0.6, tl.srt = 45, tl.col = "black")

and this will always work , all we need to is just supply the visualisation with any numeric fields we wish to analyse. However note that to use the corrplot library we would needinstall it first with install.packages(“corrplot”) either form R studio or Visual Studio on the workstation or VM we are working on.

Conclusion

This lab was intended to introduce you to the basic concepts of how to use R in conjunction with SQL Server 2016 and Power BI.

**Next Steps:**

* R resources on a page <http://revolutionanalytics.com/r-language-resources>

Notes

In order to setup a SQL Server 2016 VM you’ll need to do the following:

1. Install SQL Server with R integration
2. Configure SQL Server to allow external scripts to be run..

Exec sp\_configure 'external scripts enabled', 1

Reconfigure with override

1. Confirm R is running OK ..

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

which will return 1 , Hello if everything is OK. If it isn’t make sure that the sql Launchpad service is running.

1. Install an R IDE (R Studio or Visual Studio)
2. Run the following to add the library packages used in this lab to the SQL Server VM:

install.packages("(ggplot2")

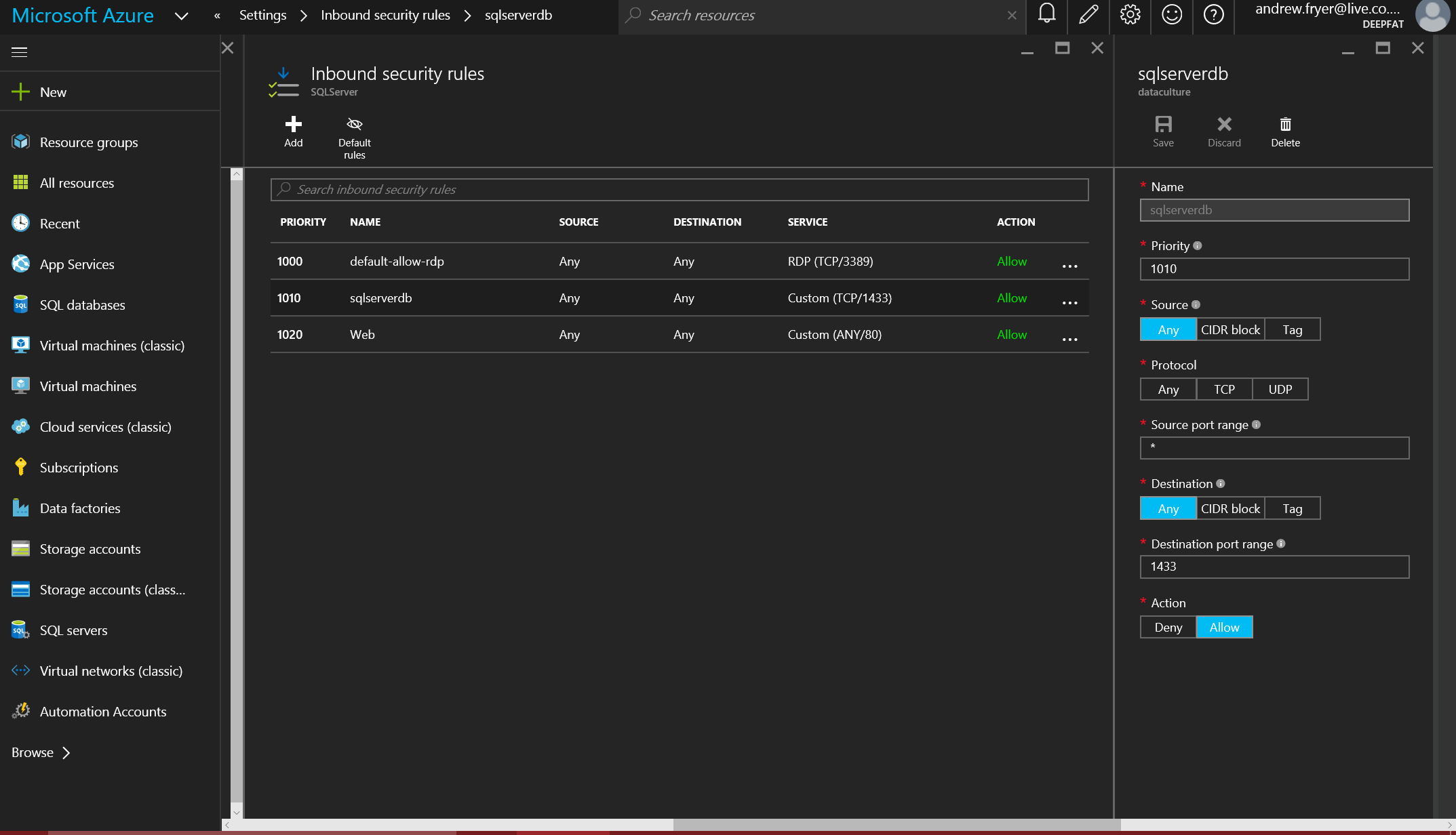
install.packages("ggthemes")

install.packages("reshape2")

1. If you want to access SQL Server on this VM from another VM or your local machine then you’ll need to do two additional steps:
   * Open port 1433 inside the VM in advanced firewall setting with PowerShell

New-NetFirewallRule -Name 'SQL Server' -DisplayName 'SQLServer' -Protocol TCP -LocalPort 1433 -Direction Inbound

* + Open up the same port to the VM in the Azure Portal. Track down the Network Security Group associated with the SQL VM and add in an additional inbound security rule to the group as shown. On the following page\;



1. For more on R Open and Enterprise server at refer to the [Cortana Intelligence blog](https://blogs.technet.microsoft.com/machinelearning/2016/01/12/making-r-the-enterprise-standard-for-cross-platform-analytics-both-on-premises-and-in-the-cloud/) [↑](#footnote-ref-2)
2. the code for this is in the advanced analytics samples for SQL server 2016 [↑](#footnote-ref-3)