2.0

Advanced Analytics with R. (February 2025).

This document provides procedures presented in the Advanced Analytics with R to be used alongside AML Transaction Monitoring Guidance.

JUBE

Amendments

Date	Author	Version	Description
17th February 2025	Richard Churchman	2.0	Updated branding to be adjacent to presentation materi-als.

Introduction

This document contains procedures to be used in combination with the Predictive Analytics training delivered at Jube. The intention is to provide a desk reference to ensure that the skills obtained in the training course by Jube, do not fade and can be used consistently in practice.

Get Datasets

Download the datasets from:

https://www.dropbox.com/scl/fi/uqegr4yjqefolaiz4mf9d/Datasets.zip?rlkey=nsri0je45c03a5tbrrpegki3w&dl=0

Get Help

Email questions to:

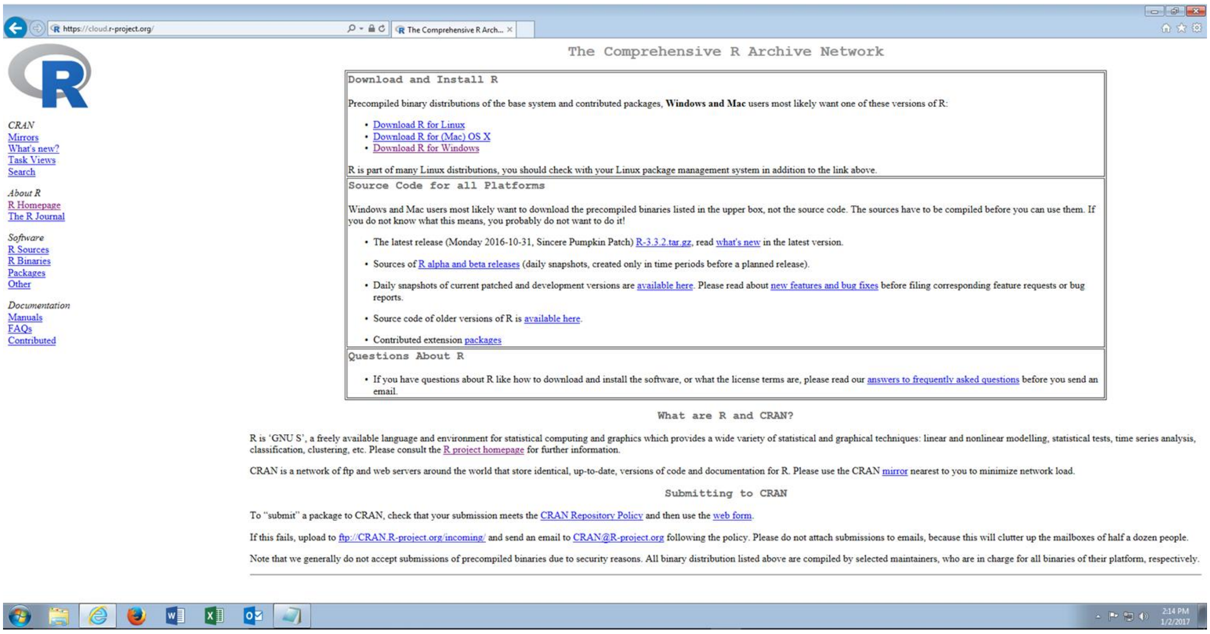
richard.churchman@jube.io

Module 2: Getting Started with R.

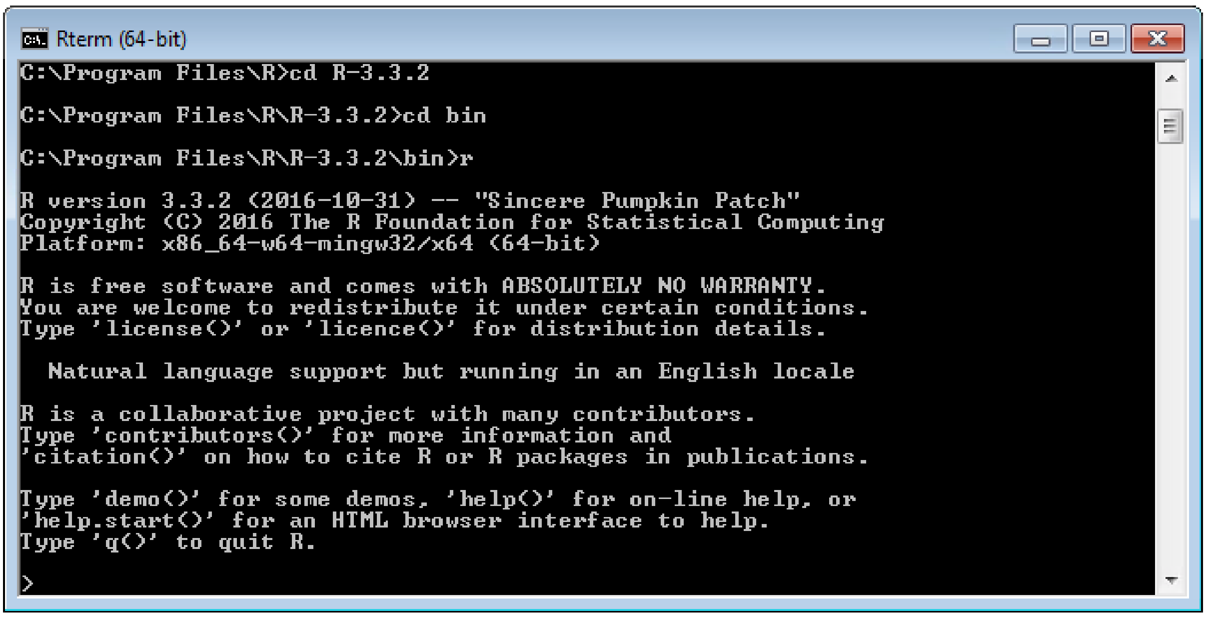
R is the software package that is the primary focus of this training. For this module, R is two separate software packages and installs.

Core R is available from https://www.r-project.org/ or by using a mirror such as http://cloud.r-project.org. Core R is created and published by the R Core Development Team. Fundamentally the view that Core R is a command line only tool, used for production deployments only, should be adopted. R Core \ R Command Line is used very little in this training course and is predominantly shown as a precursor to RStudio Console.

Once R Core is downloaded and installed, the command line application is available in C:\Program Files\R\R-3.3.2\bin\titled R.exe although navigation to and invocation of the application is detailed in procedure 1.

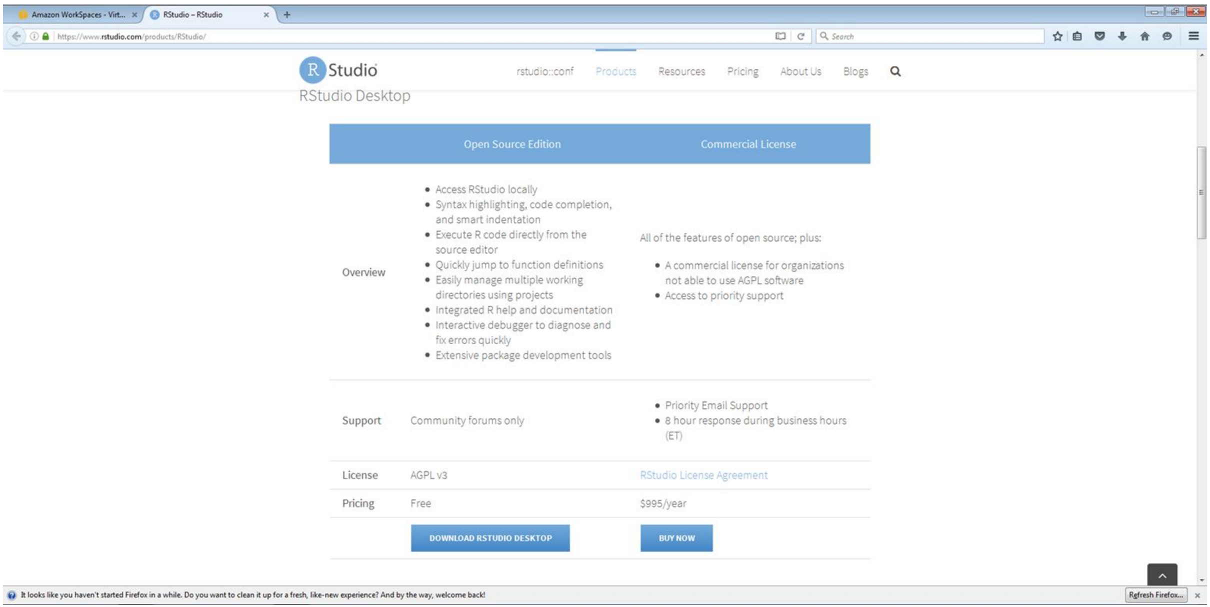


In this example, the latest version of R for Windows has been installed with the default settings.

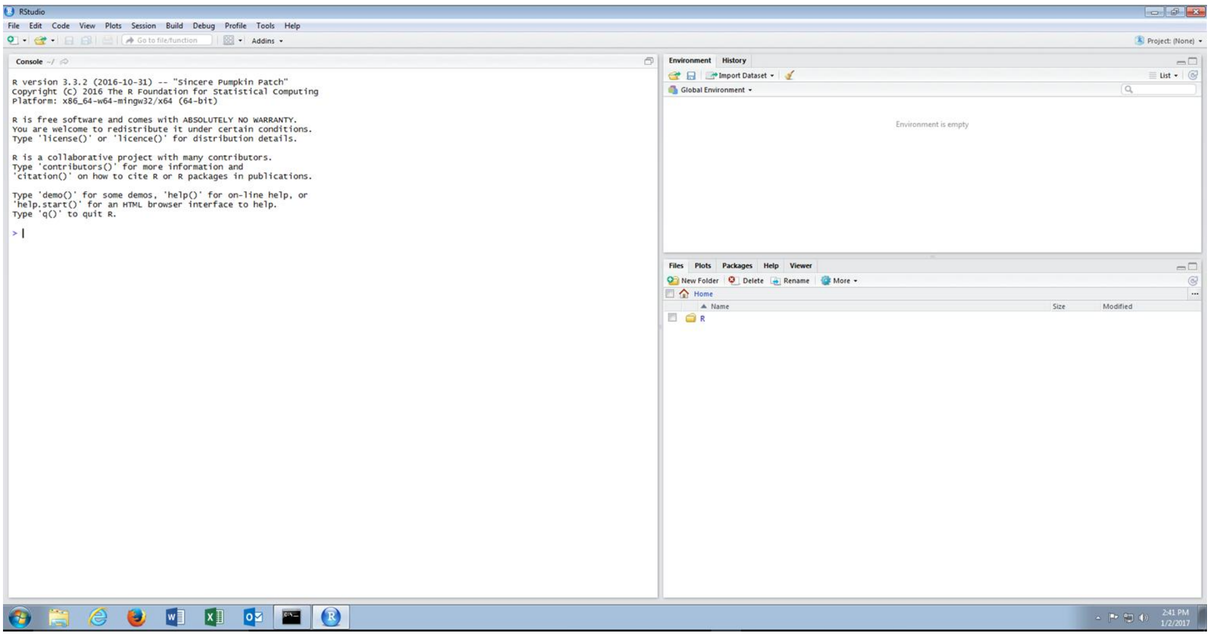


RStudio is a feature rich Integrated Development Environment (so-called IDE) that improves productivity in creating R Scripts, although in production the execution of these scripts might well fall to the core installation.

The software can be downloaded from https://www.rstudio.com/products/RStudio/ and is free, although there are commercial editions.



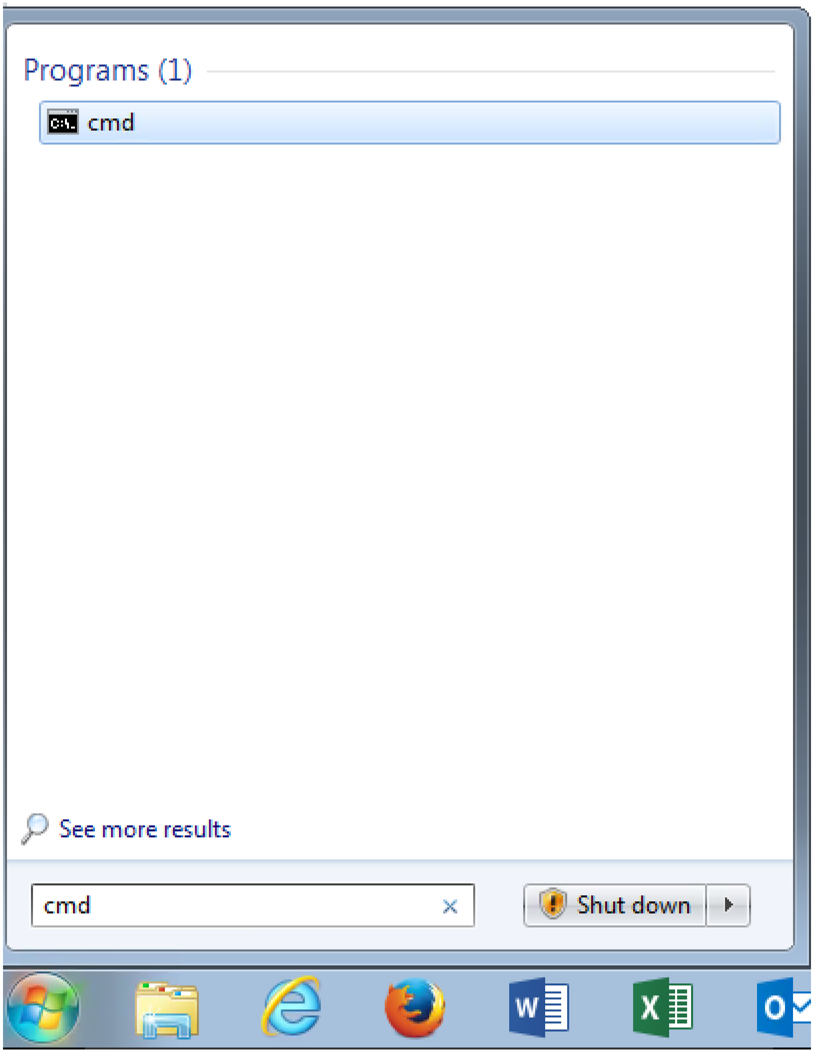
As with R Core, the defaults have been left unchanged during the installation.



Procedure 1: Navigate to and launch the R command line.

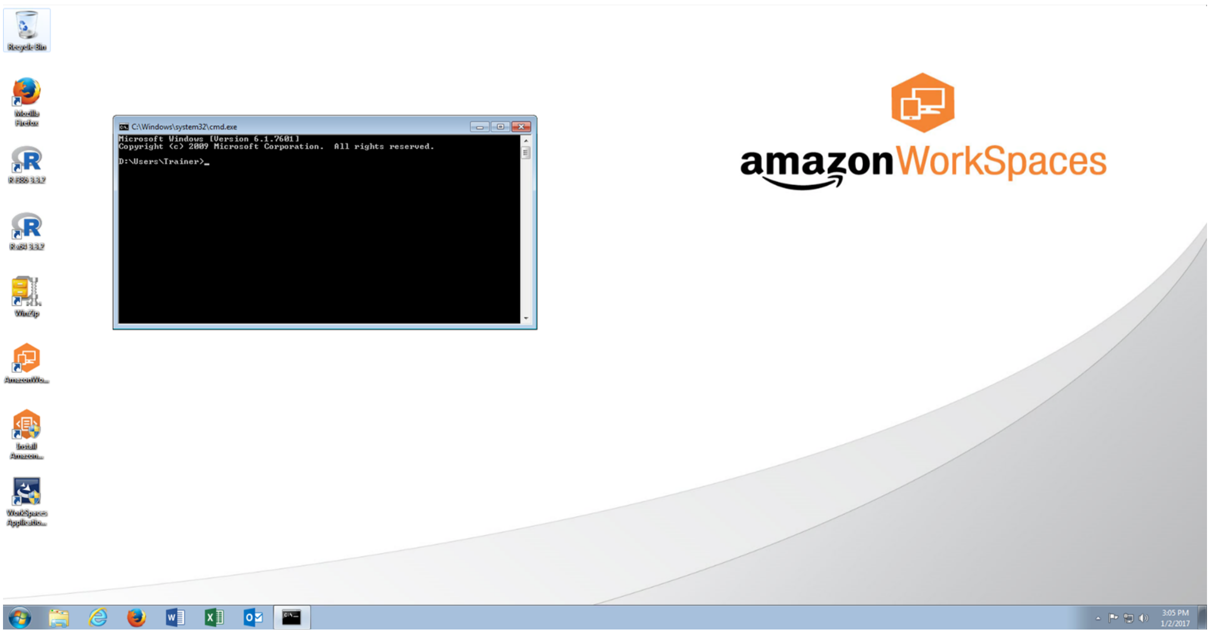
To launch the R Core Command Line software, start by launching the command prompt. The quickest way to launch the command prompt is to click the Start button firstly:

Then in the run \ search bar type CMD, which will suggest the appropriate application:



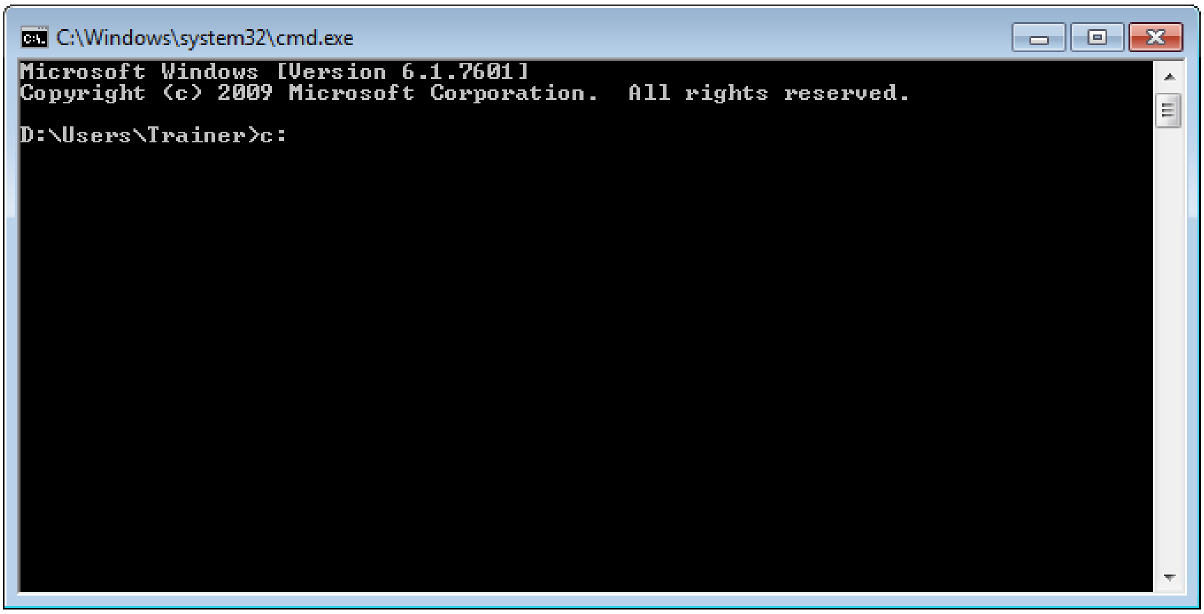
Click on, rather run, the application:

JUBE

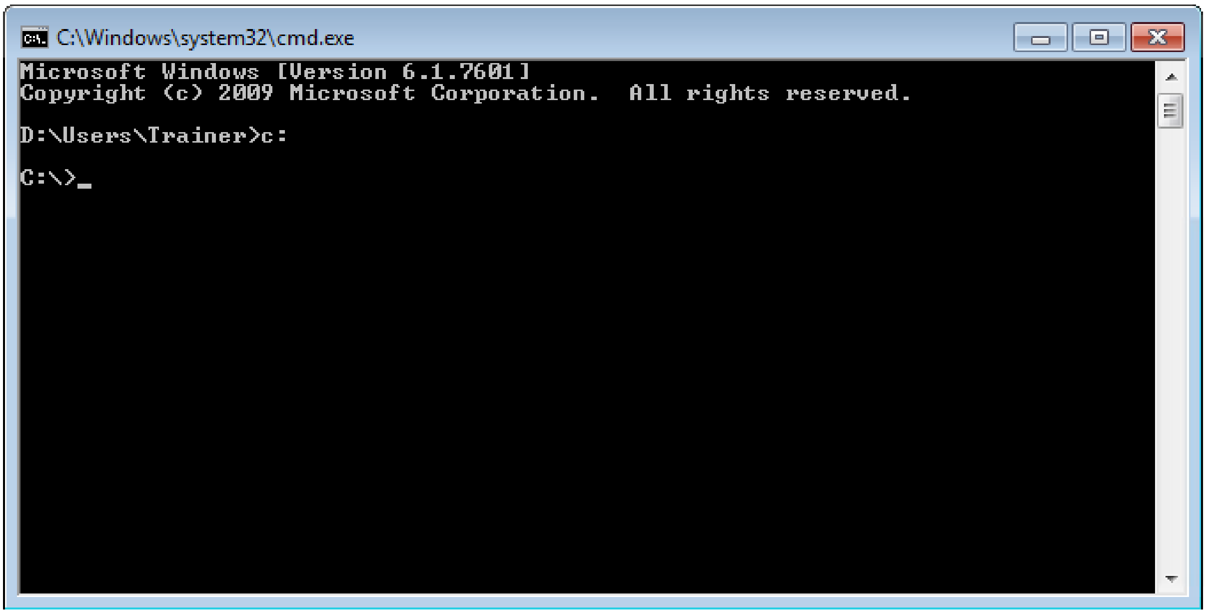


It is unlikely that the Command Prompt will be in the correct directory to run R. Switch to the C:, which is where all installed programs tend to reside, by typing:

c:

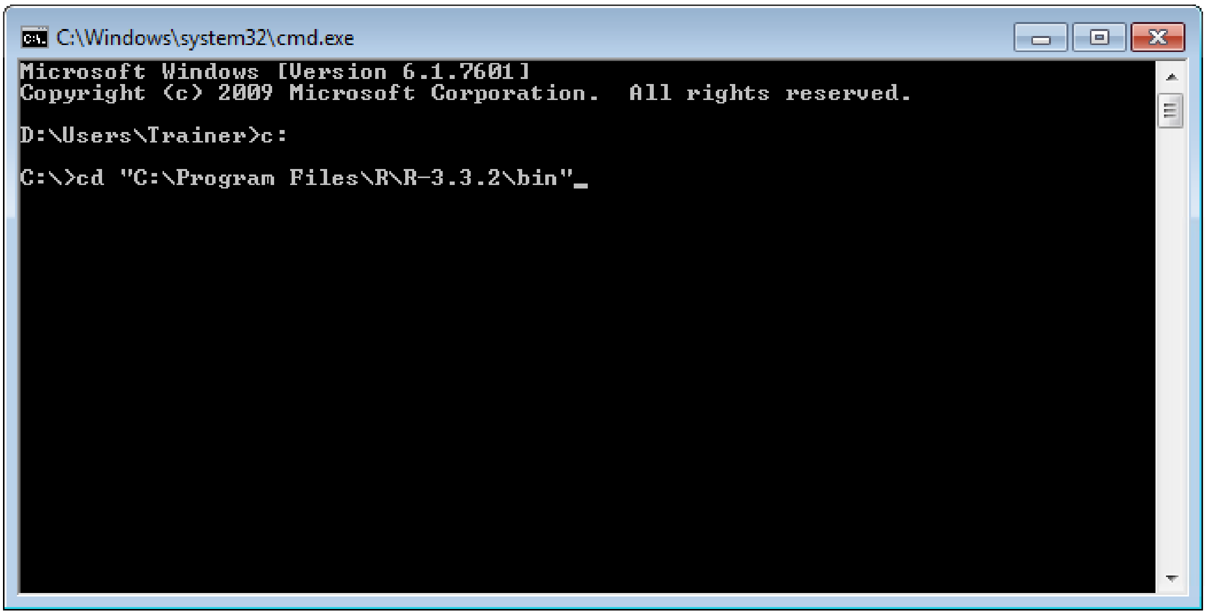


Press the Enter key to make the drive change:

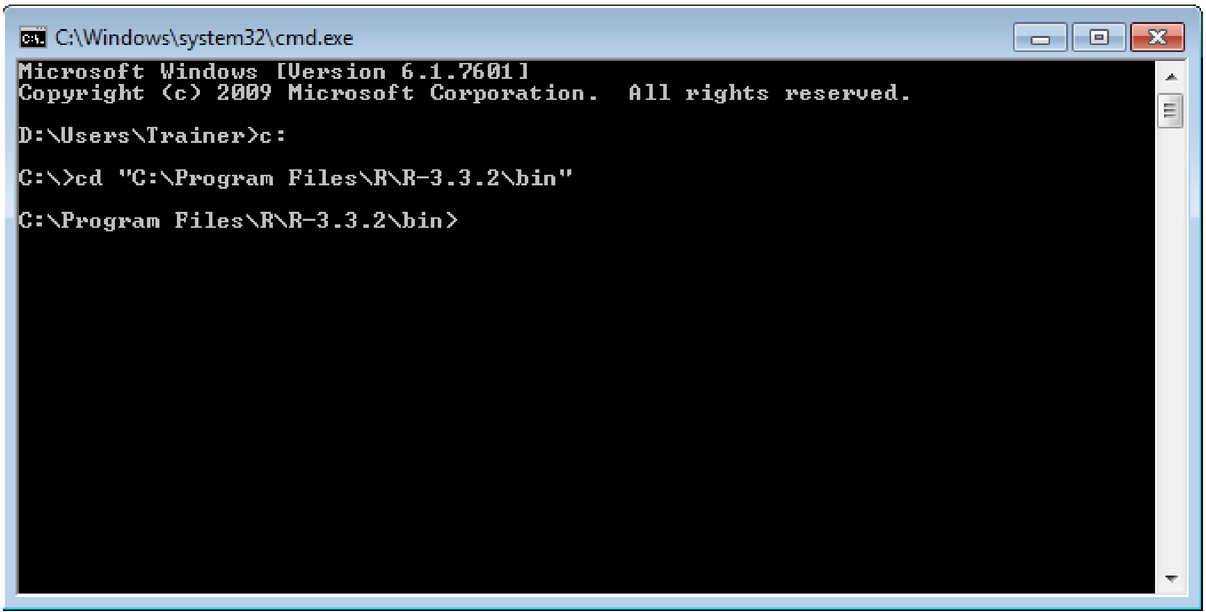


To navigate to the directory containing the R command line type:

cd "C:\Program Files\R\R-3.3.2\bin"

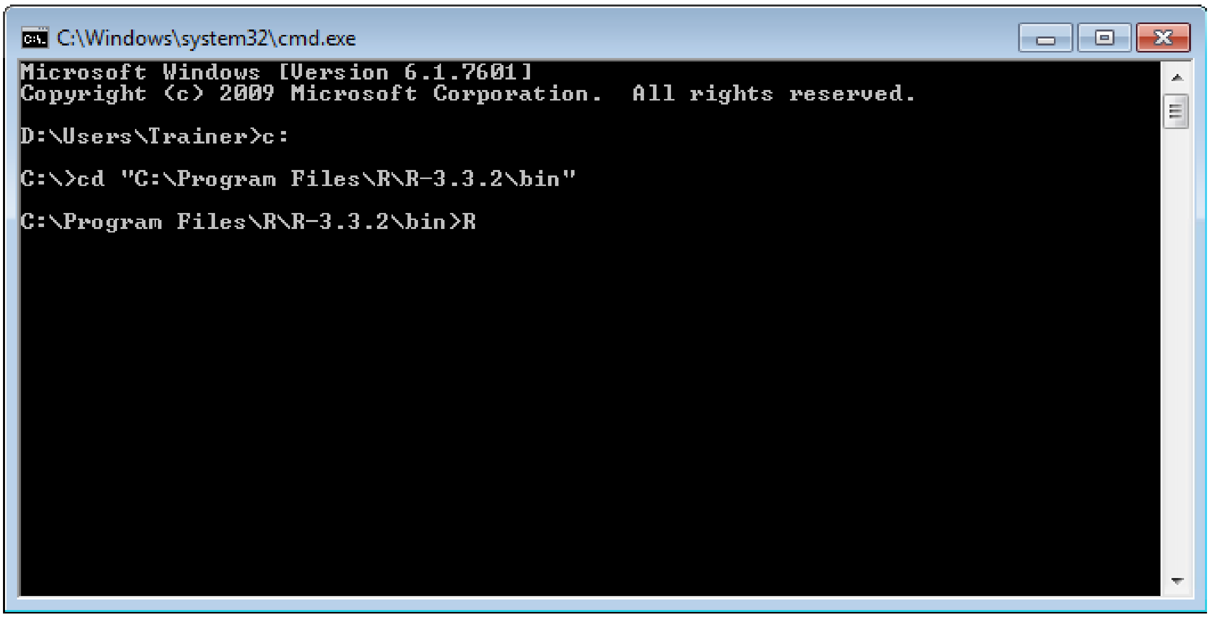


Commit the drive by change pressing the Enter key:



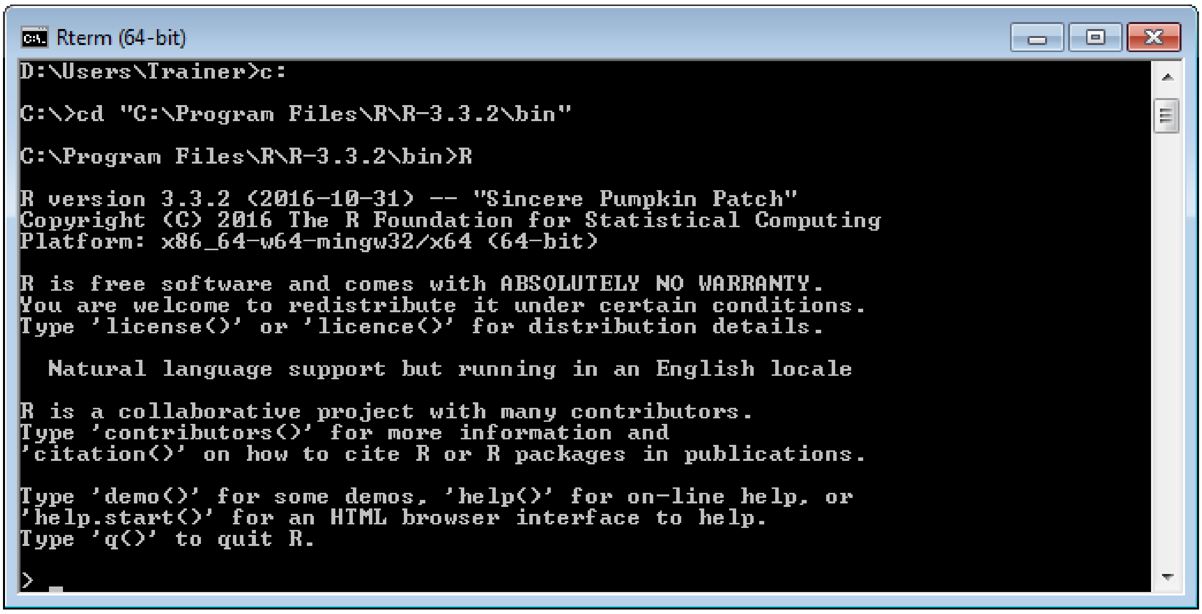
To launch the R console application type:

R



Invoke R Core by pressing the Enter key:

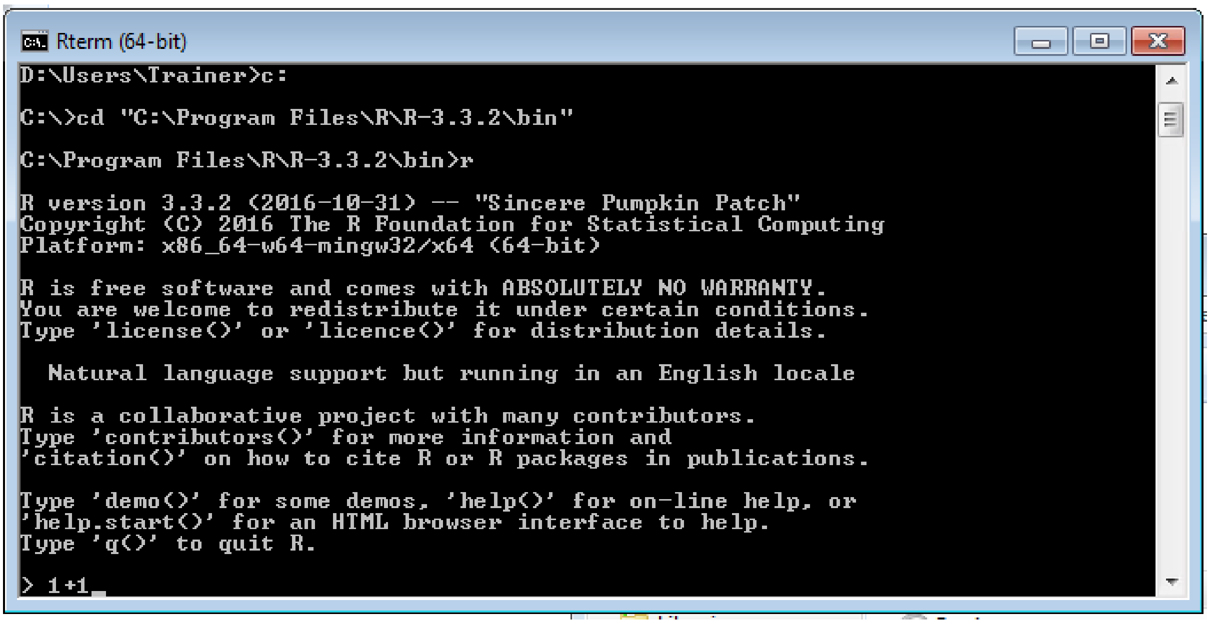
Upon sucessful launch of the R Core Command Line Interface, introductory text will be displayed with a chevron (i.e >) denoting the command line input awaiting with a flashing cursor:



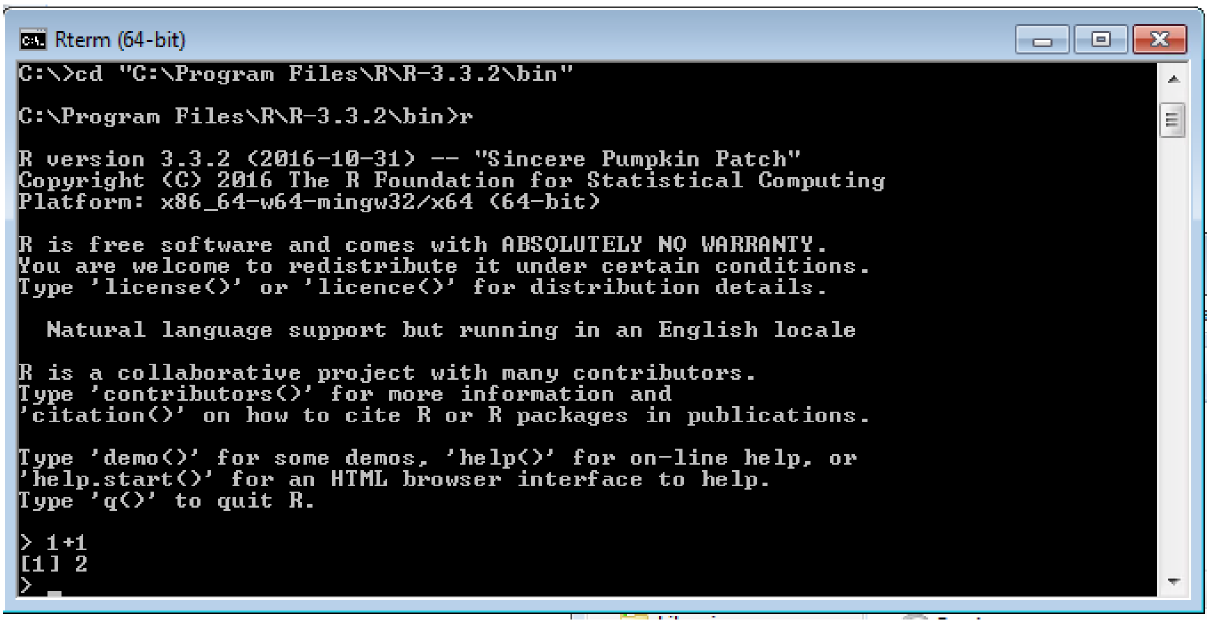
Procedure 2: Issue commands to the R Console.

R is an interpreted language for mathematical and statistical computing. R processes as script, line by line. In this example the sum of 1 + 1 will be returned, which will of course be 2. To perform such a calculation type:

1+1



Press the Enter key to commit and execute the line of script:

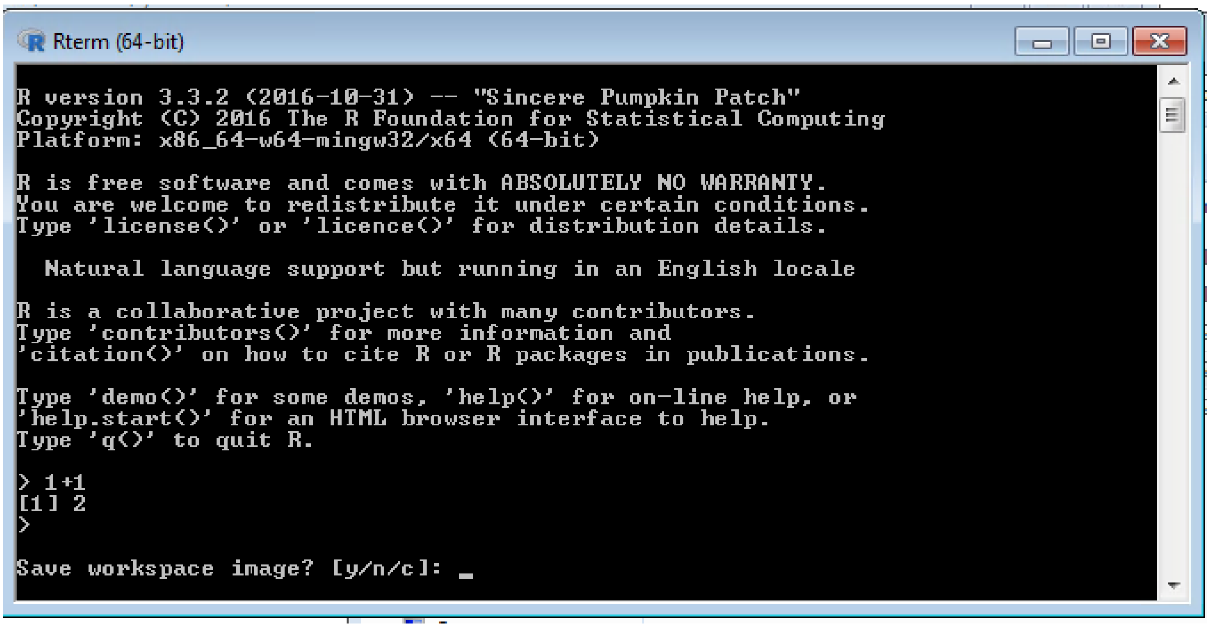


It can be seen that a line has been returned showing [1] 2, where [1] is the position in the result vector, where 2 is the actual value returned from the line of script. The mathematical operators (in this case +) are much the same as Excel:

- + Addition.
- - Subtract
- / Divide
- * Multiply

This procedure has shown a simple line of script being written, executed and returned by R. Although rudimentary, it is an R program.

To exit the R console, hold down the CTRL key and the D key:



There are three options presented when exiting the R console:

- y: Save the workspace image for reloading. This will keep everything in the current session.

- n: Clear the workspace so that the next time r is loaded it will be afresh.
- c: Cancel and go back to the workspace.

In this example, type:

y

```
Rterm (64-bit)
C:\Program Files\R\R-3.3.2\bin>r

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
[1] 2
>

Save workspace image? [y/n/c]: y
```

Press the Enter key to commit the command:

```
C:\Windows\system32\cmd.exe
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
[1] 2
>

Save workspace image? [y/n/c]: y
Unable to open .Rhistory

C:\Program Files\R\R-3.3.2\bin>_
```

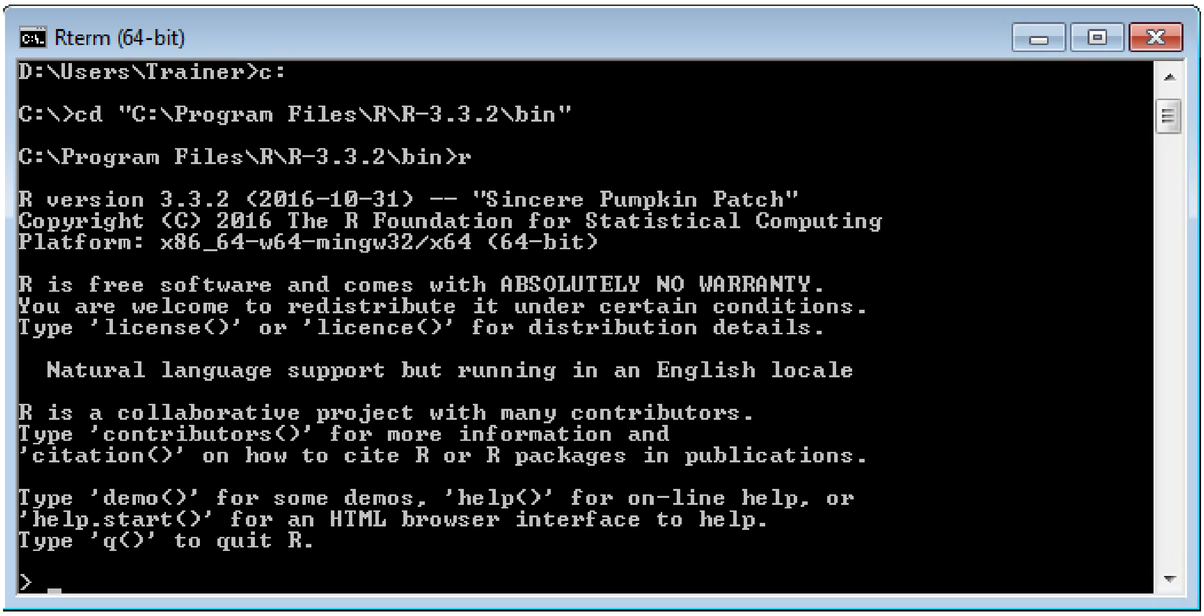
Notice that an error was returned 'Unable to open .Rhistory'. The error is created as the operating system will not allow the user to write to the same directory as R is running, which introduces the concept of working directories, as follows.

Procedure 3: Set a Working Directory.

A working directory is where R will look for files during a session. The files may be the R session, or in subsequent procedures it will be data to be imported and data saved as the result of processing.

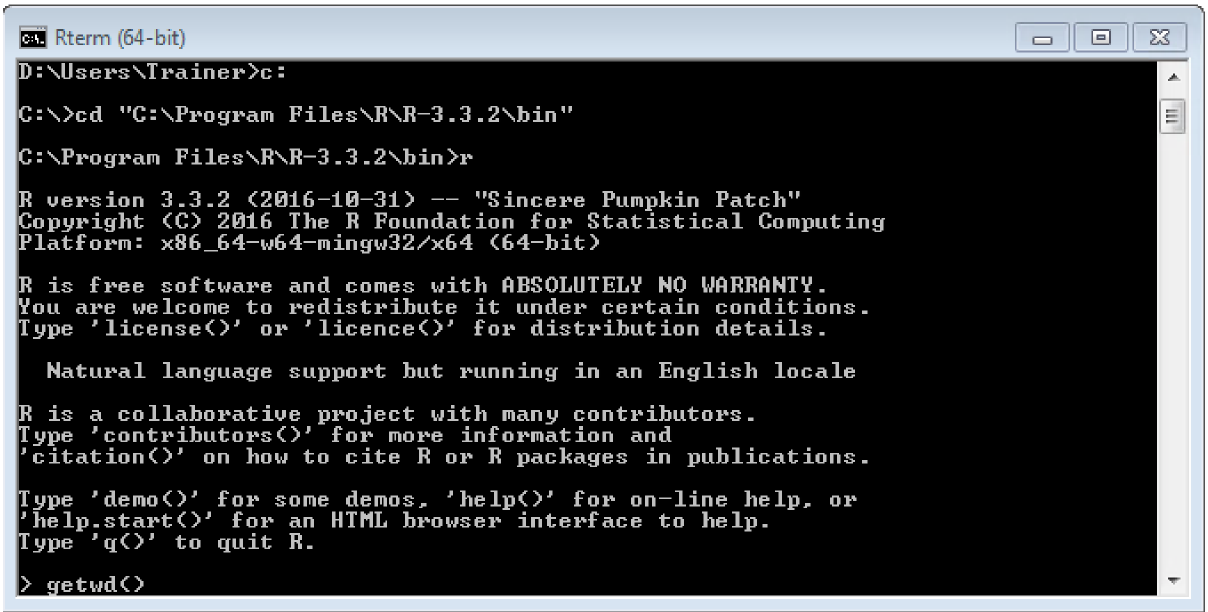
In proecedure 2, it was observed that there was a failure when saving the R history, owing to the working directory not being set (rather set incorrectly). It follows that the working directory need be set.

Start by executing procedure 1 to load the R console.

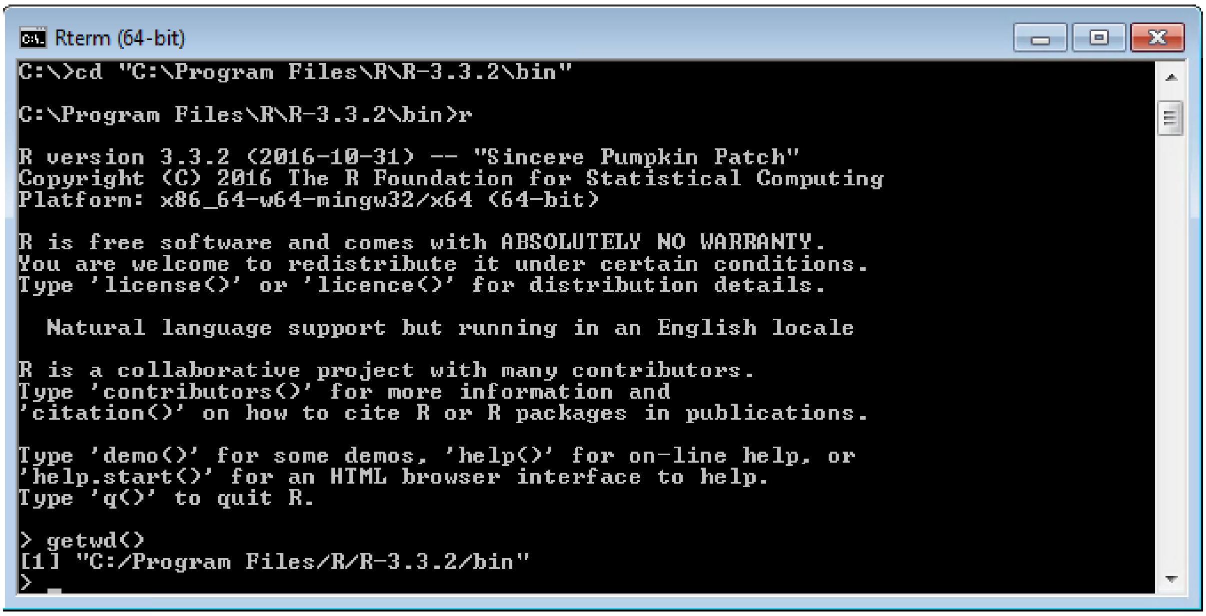


To identify the current working directory use the getwd() function, type the script line:

getwd()

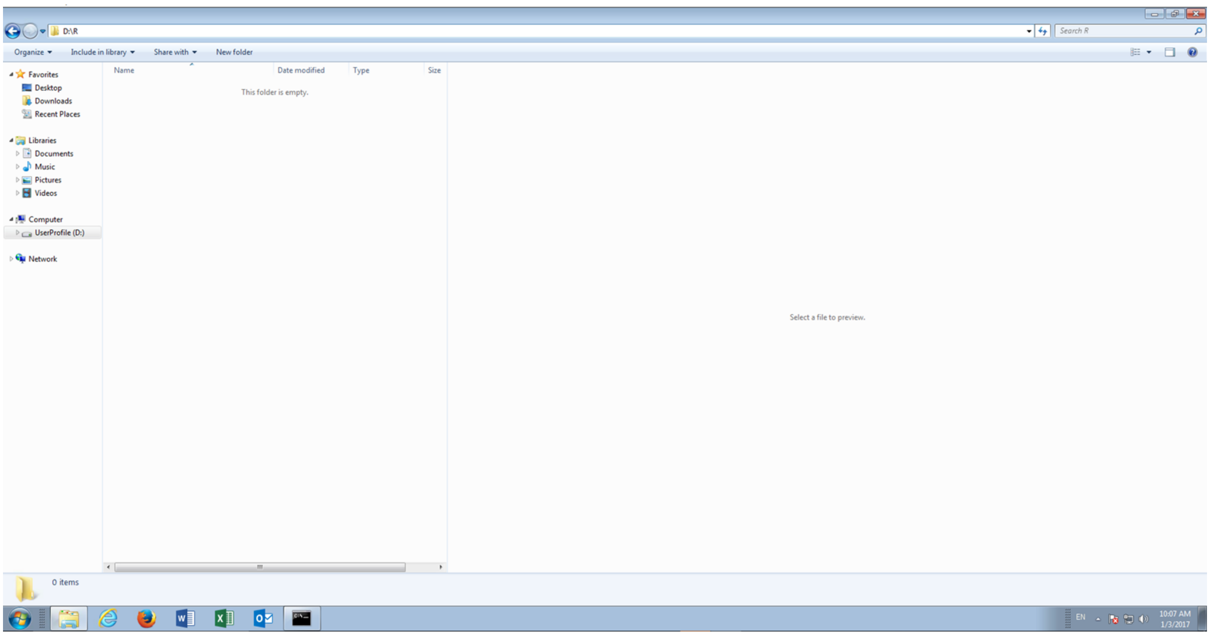


Execute the command by pressing the Enter key:



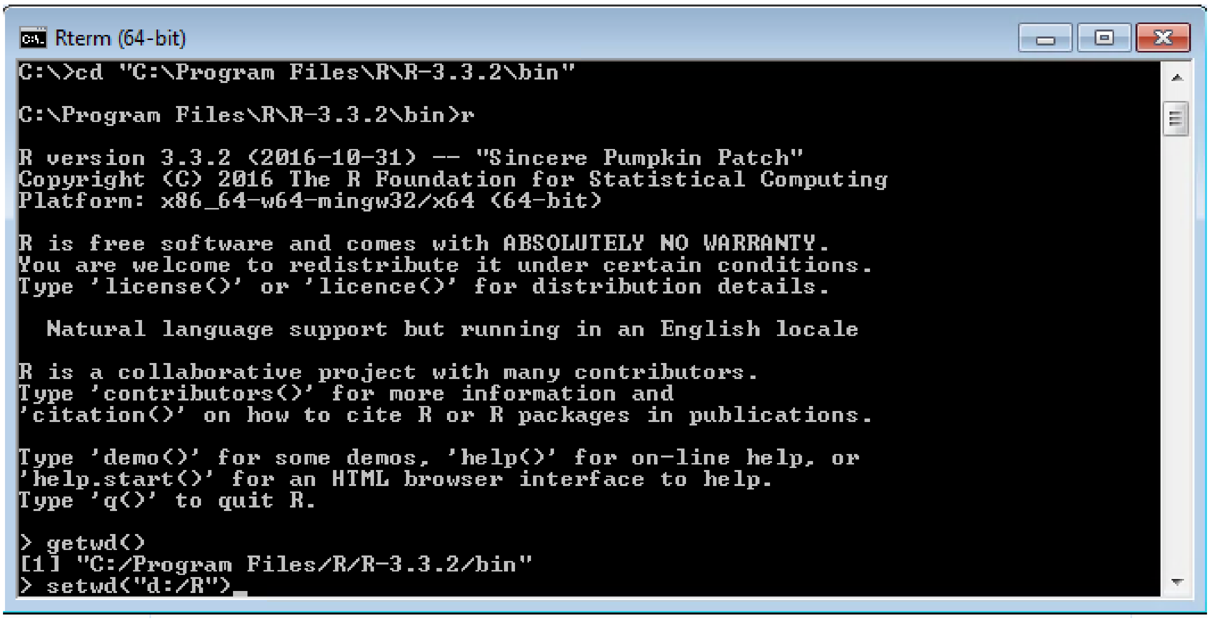
The current working directory, which is the directory containing the executable, is returned. Saving files to the same directory as the R software is not desirable, quite beyond it causing errors, and as such, this should be changed to an appropriate directory.

Create a directory to be used throughput these procedures. In this case the files will be saved to the d:\ in a directory called R.



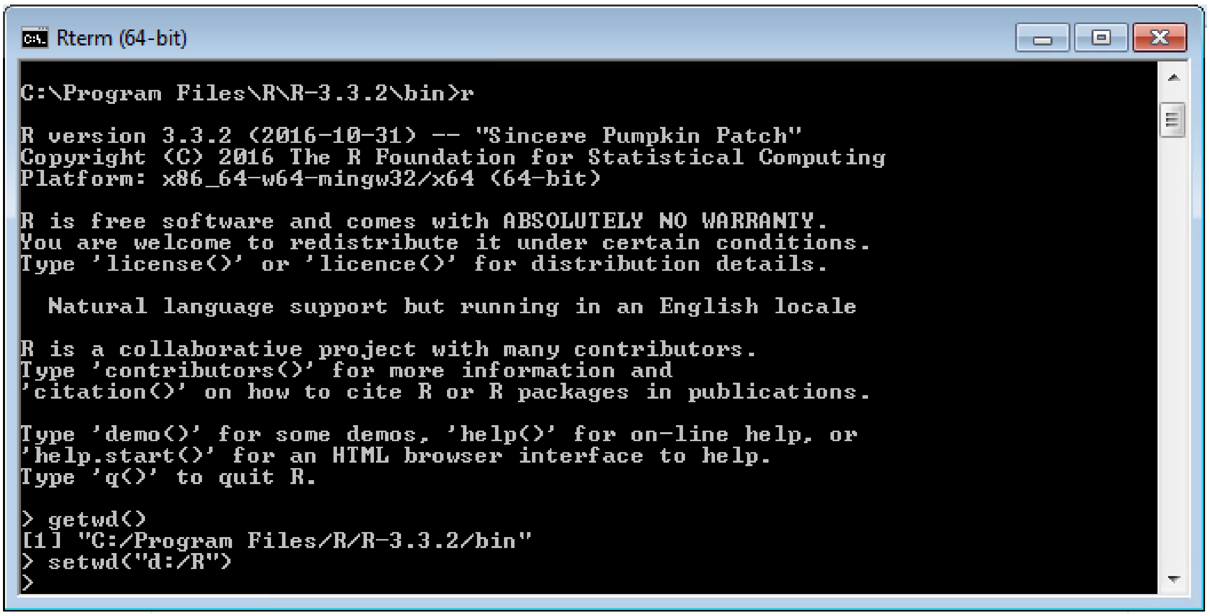
To set this as the working directory in R use the setwd() function with the directory in quotation marks, type:

setwd("d:/R")



```
Rterm (64-bit)
C:\>cd "C:\Program Files\R\R-3.3.2\bin"

C:\Program Files\R\R-3.3.2\bin>r

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "C:/Program Files/R/R-3.3.2/bin"
> setwd("d:/R")
```

Press the Enter key to process the line of script:



```
Rterm (64-bit)

C:\Program Files\R\R-3.3.2\bin>r

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "C:/Program Files/R/R-3.3.2/bin"
> setwd("d:/R")
>
```

The absence of any error message confirms that the working directory has been changed, although this can be affirmed by executing the getwd() function:

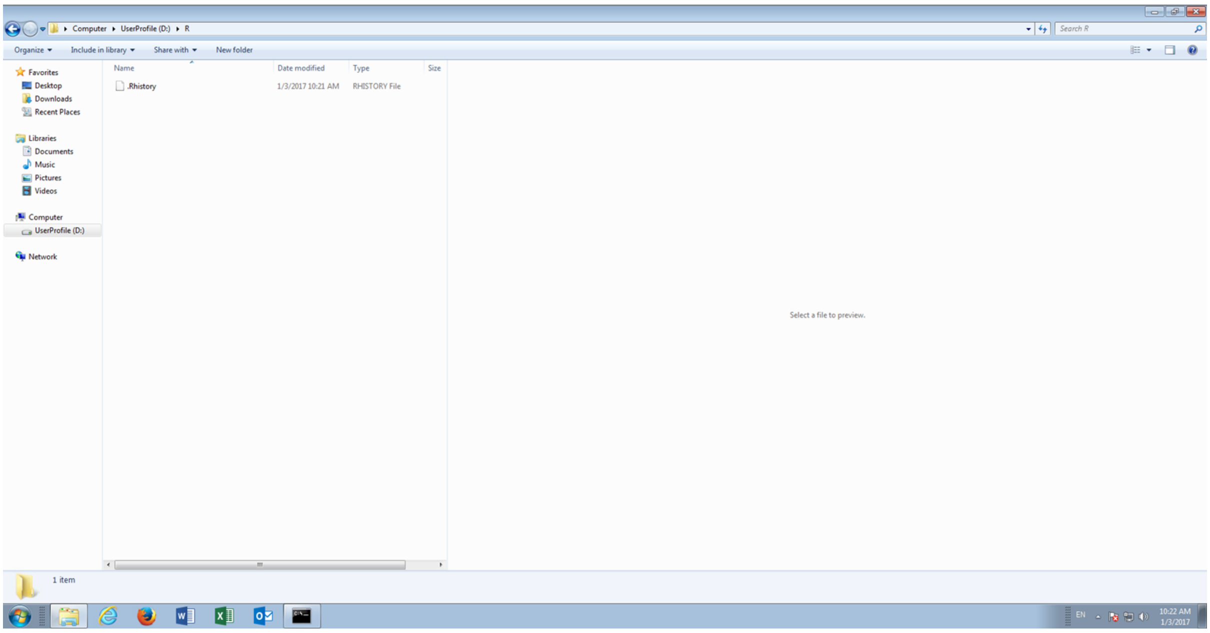
```
Rterm (64-bit)

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "C:/Program Files/R/R-3.3.2/bin"
> setwd("d:/R")
> getwd()
[1] "d:/R"
> _
```

The working directory is now set to d:\r.

If R is exited, and y is selected to save, it can be observed that there were no errors:

```
C:\Windows\system32\cmd.exe

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "C:/Program Files/R/R-3.3.2/bin"
> setwd("d:/R")
> getwd()
[1] "d:/R"
>

Save workspace image? [y/n/c]: y

C:\Program Files\R\R-3.3.2\bin>
```

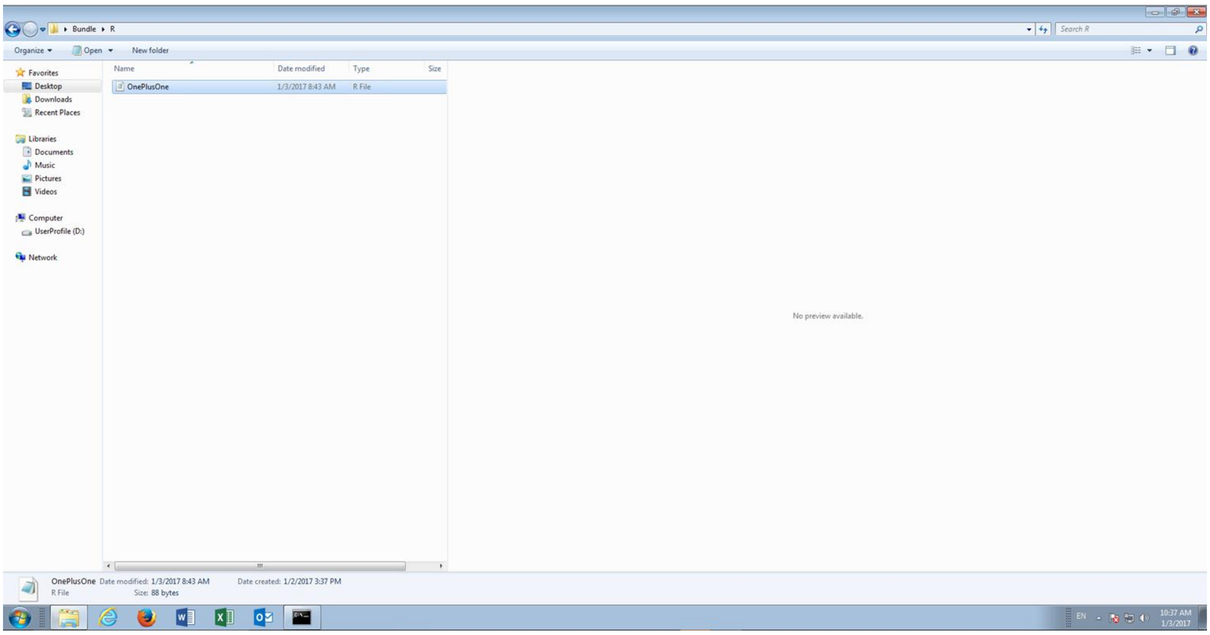
Furthermore, it can be seen that the .RHistory file has been saved to the working directory:



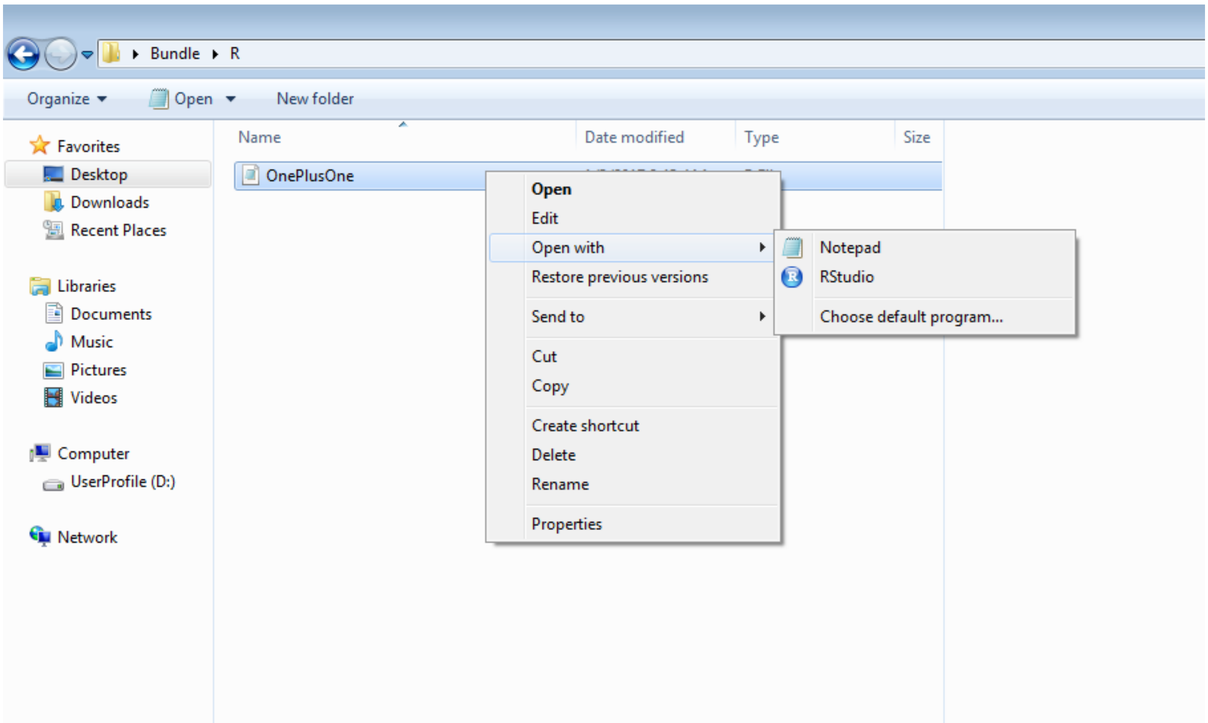
Procedure 4: Run a script from the R command line.

The procedures presented thus far have used the R Console to directly process commands line by line, requiring the Enter key to be pressed to execute. An alternative means to execute R commands is a script execution approach, where each script line is presented as the line of a text file.

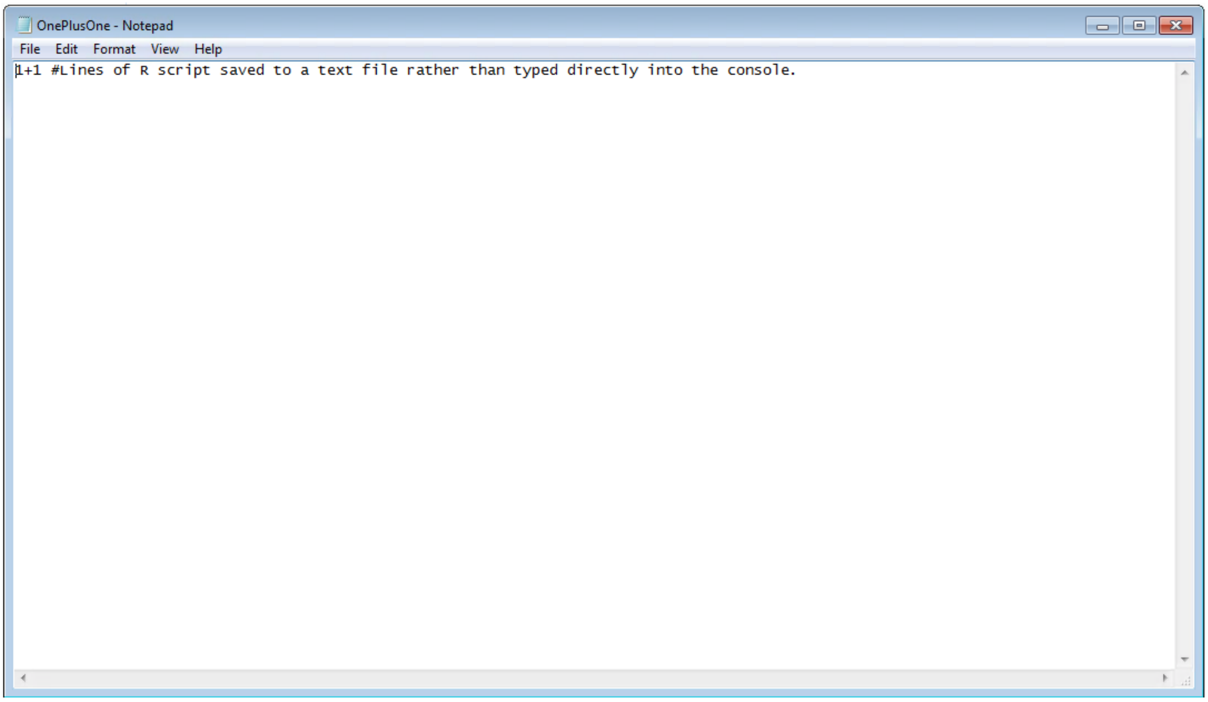
In Windows Explorer navigate the directory Bundle\R\:



In the directory Bundle\R, there is a file called OnePlusOne.r. Right click on this file:



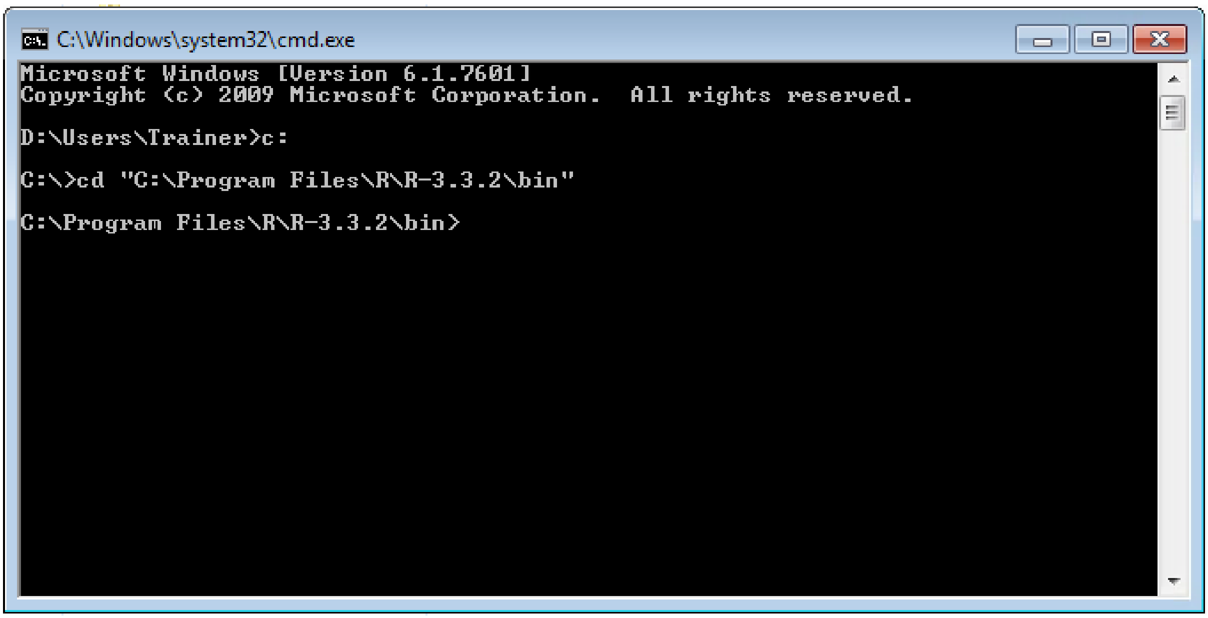
For the time being click on Notepad to open the file:



Inside the text file it can see seem that the same command that was executed in procedure 1 is present as a line in the text file. Notice also the presence of a hash tag after the command, which is a comment whereby everything after the hash (to the right of) is ignored.

For the purposes of this procedure, close Notepad, as it is purely to illustrate that the contents of the file are the same as would be entered directly into the R console.

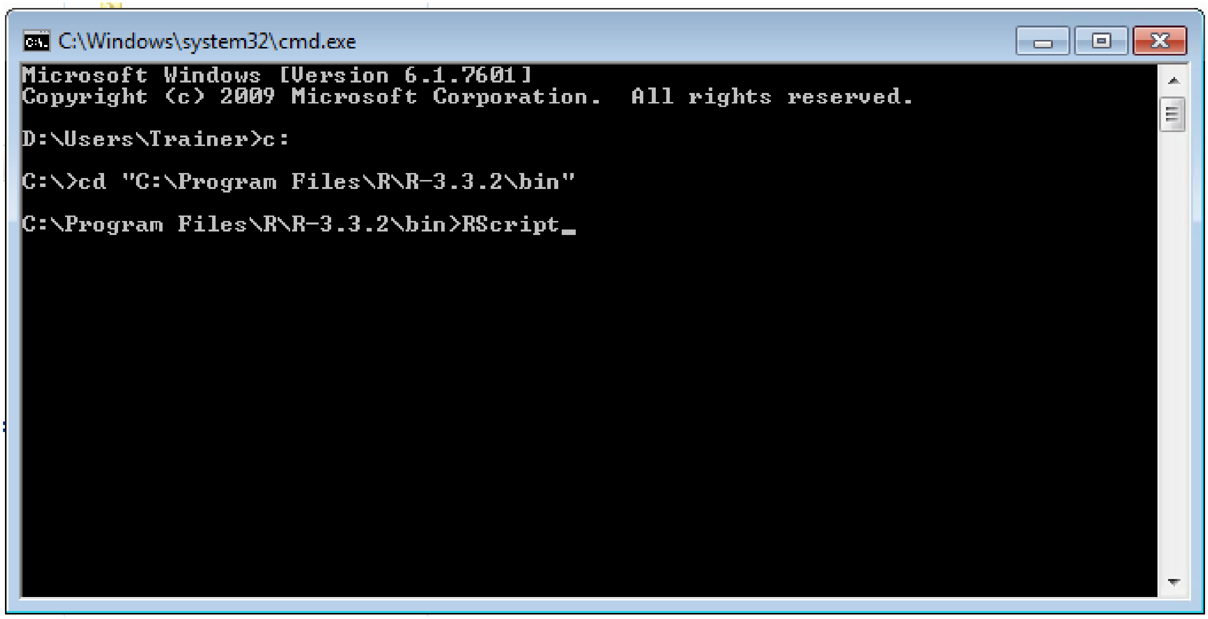
Open the command prompt and navigate to the R directory as described in procedure 1, although do not load R.exe instead this procedure uses RScript.exe:



The RScript program exists for the purposes of executing a series of R script lines rather than requiring a command to be entered one by one into the console for interpretation by R.

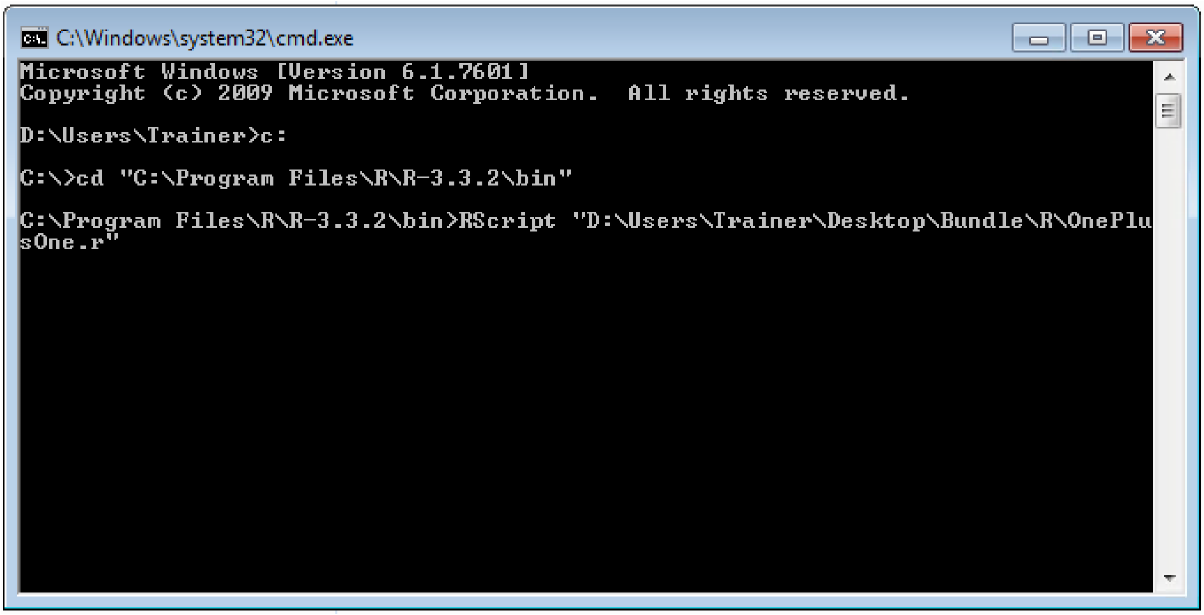
To execute the script OnePlusOne.r, start by typing:

RScript



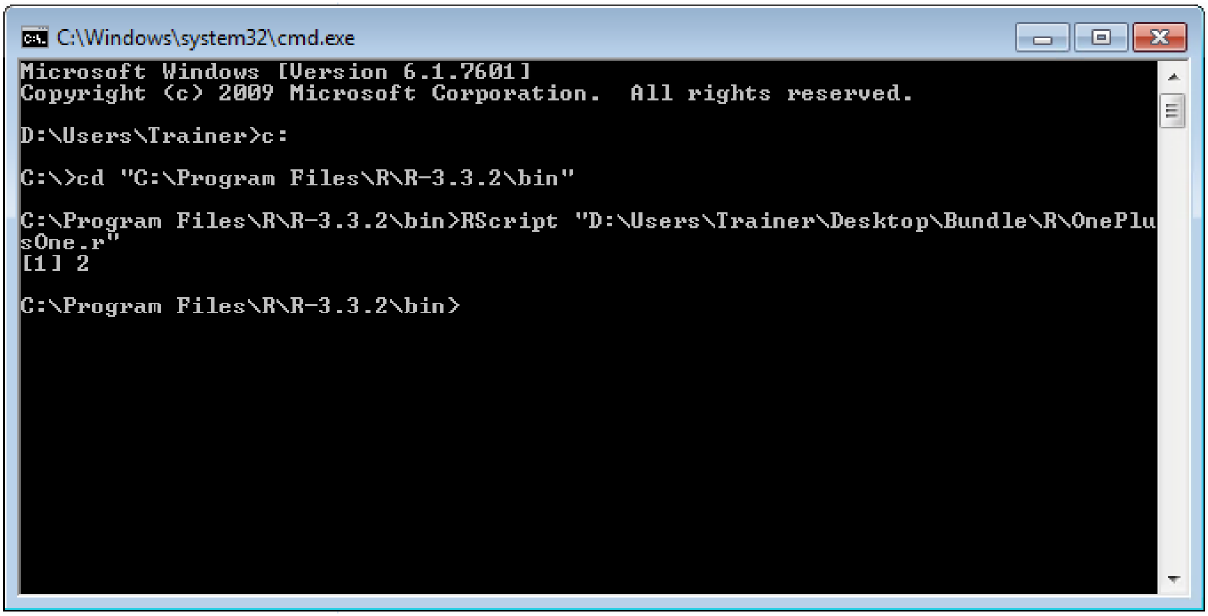
Followed by the name of the R script to execute, which is in this case Bundle\R\OnePlusOne.r:

RScript "D:\Users\Trainer\Desktop\Bundle\R\OnePlusOne.r"



Notice that the structure is the executable, RScript.exe, followed by the directory and file name of the script within double quotations.

Press Enter to launch the RScript.exe program with the script passed as an argument:

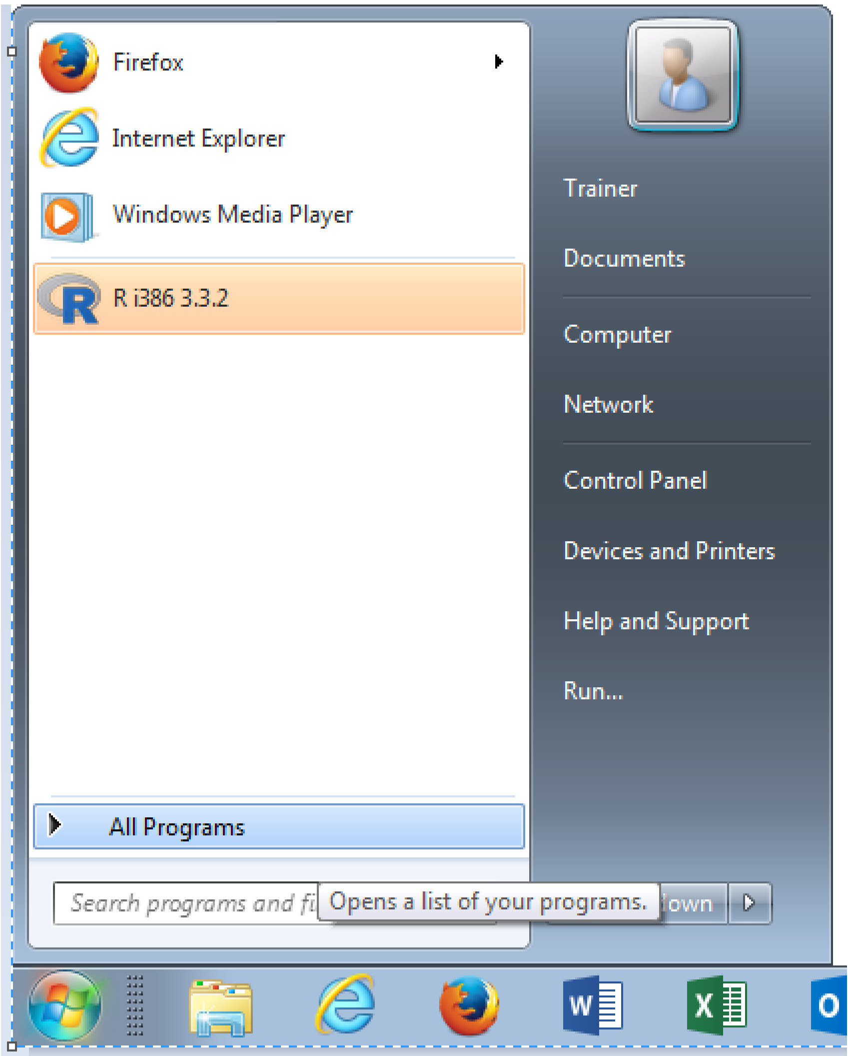


Once completed, RScript will return to the command prompt. It can be seen that the output to the command line is the same as that observed in Procedure 1.

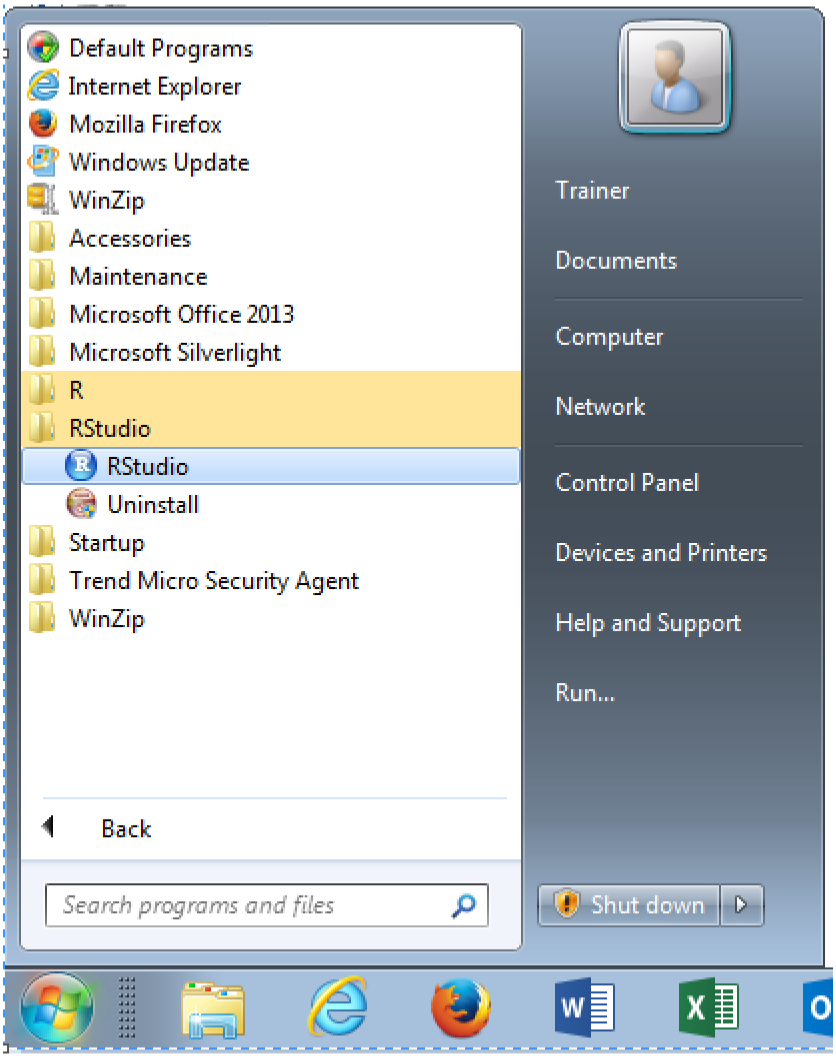
Two means of interacting with R have now been put forward, the first being the entry of command script into the R Console with the second being the staging of those commands in a text file with a view to invoking these commands in RScript.exe.

Procedure 5: Launching R Studio.

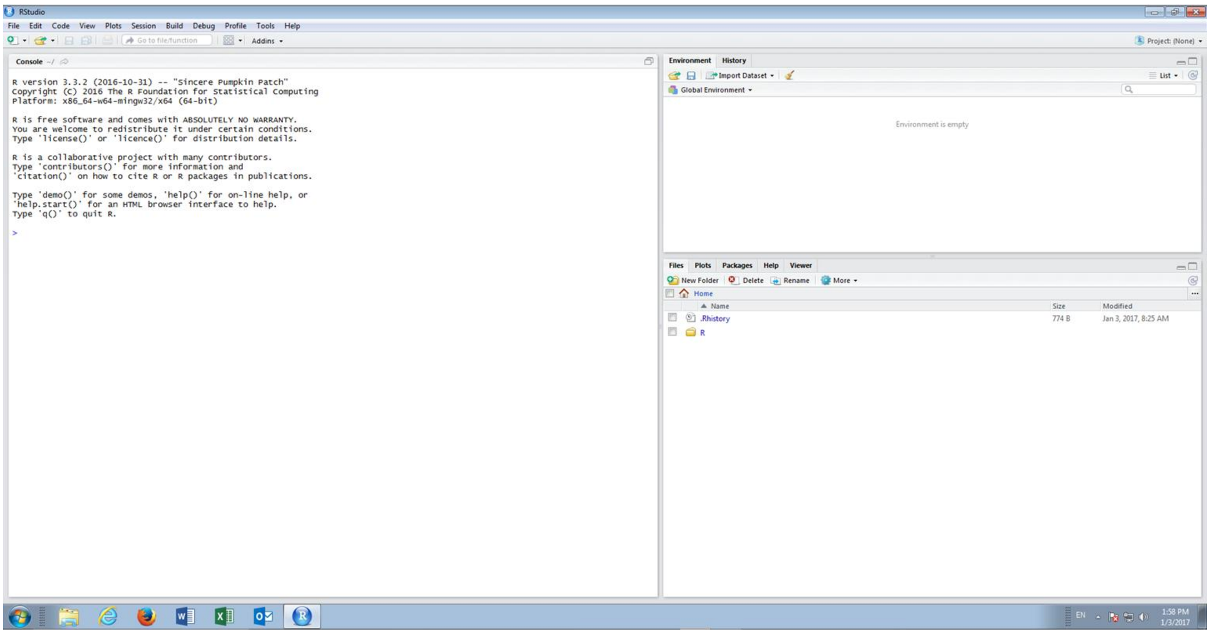
RStudio is distinct from R Core and conceptually it should be viewed that RStudio overlays RCore (although they are independent installations of R in actually). To launch RStudio, navigate to and click the Start button, then navigate to All Programs:



Expanding All Programs, navigate to the RStudio folder:

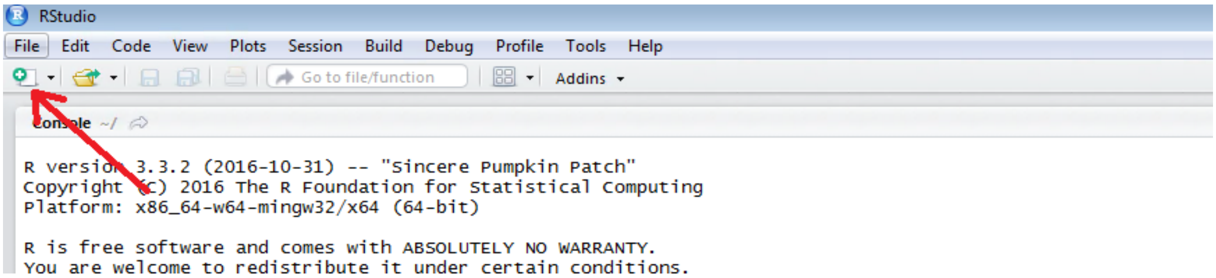


Click on the application RStudio to launch:

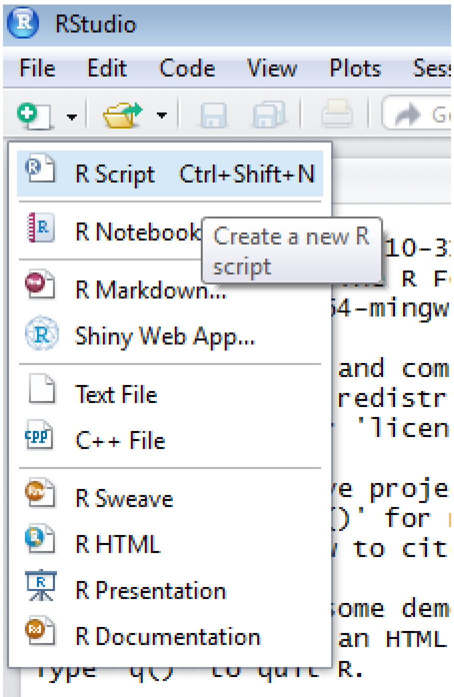


For all procedures that follow, using RStudio, a script active, console passive approach will be taken.

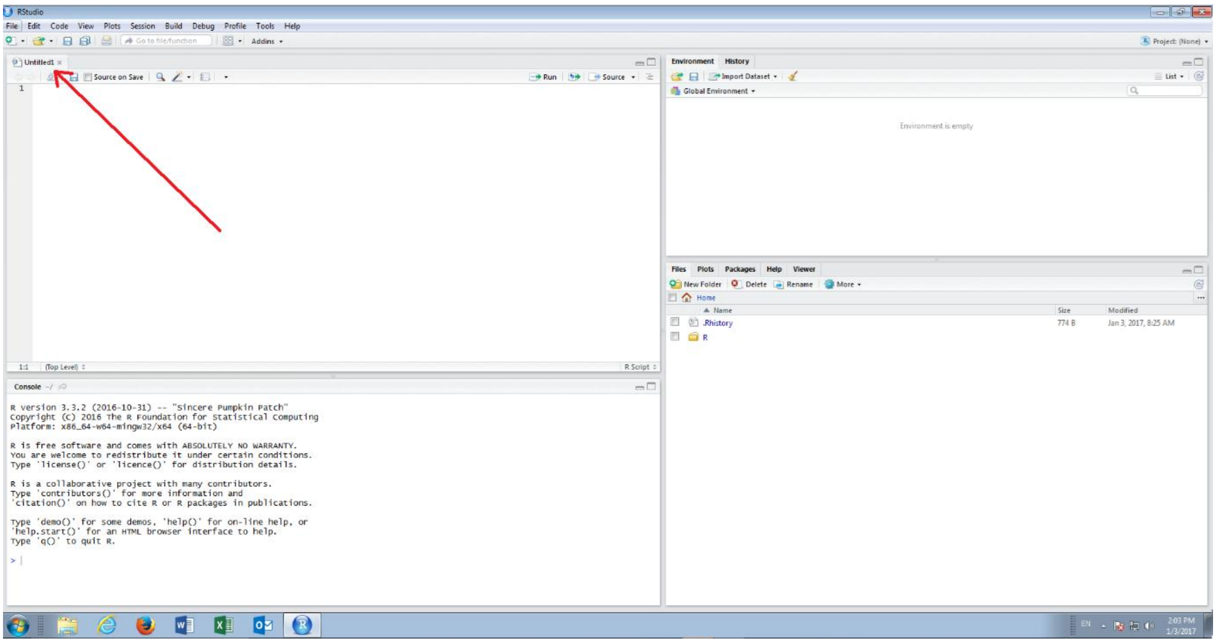
To create a new script, that will be the target for all R Console interactions, click the New Script button in the top left-hand corner of RStudio, under File:



In the sub menu, click the first option titled 'R Script':



A new, empty, script will be loaded:

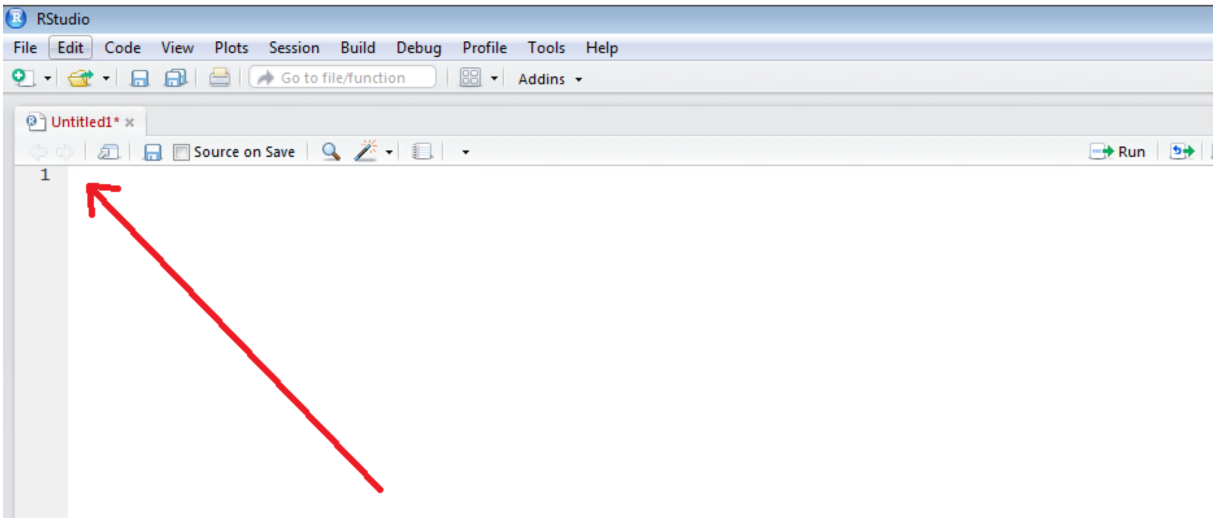


A script will be the focus of all attention, and a user will be active in the script window only, leaving the console alone. No command is ever entered directly into the console.

Procedure 6: Identify Packages Installed.

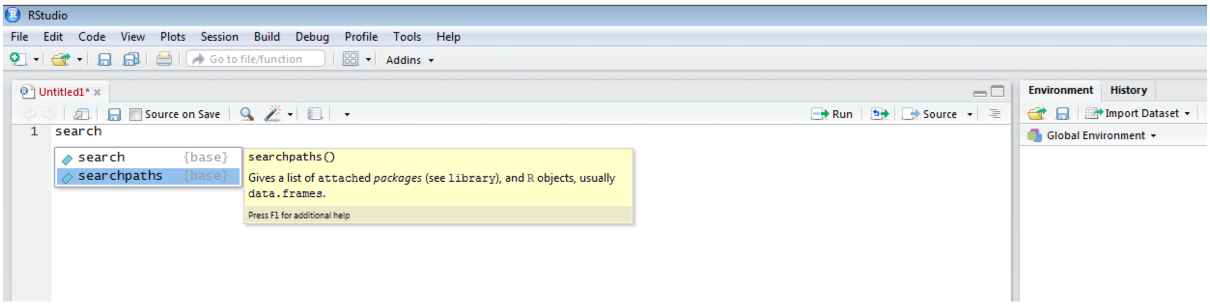
Packages are a collection of files contained in the runtime directory of R. The runtime directory, where the packages are installed, is know as the search path.

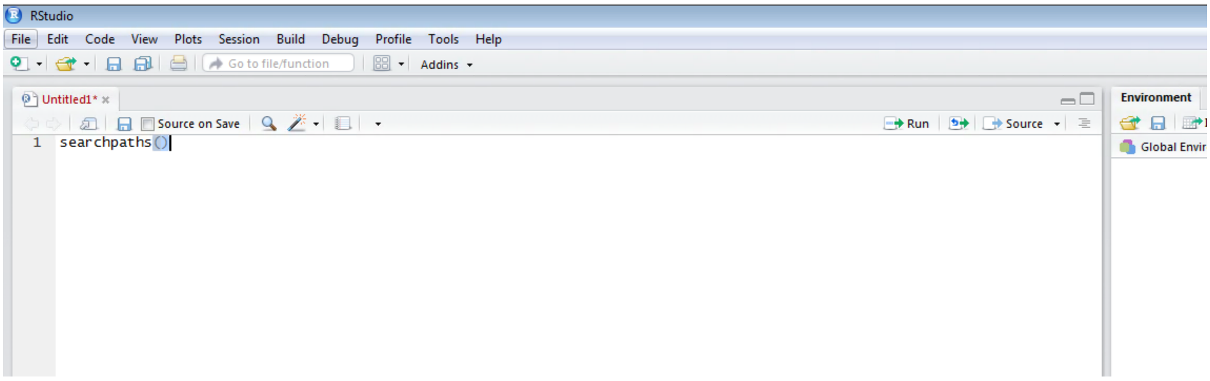
To get a picture of the pakages installed start by setting focus in the script window by clicking in the pane in the top top left hand corner:



To view the physical location of the packages type:

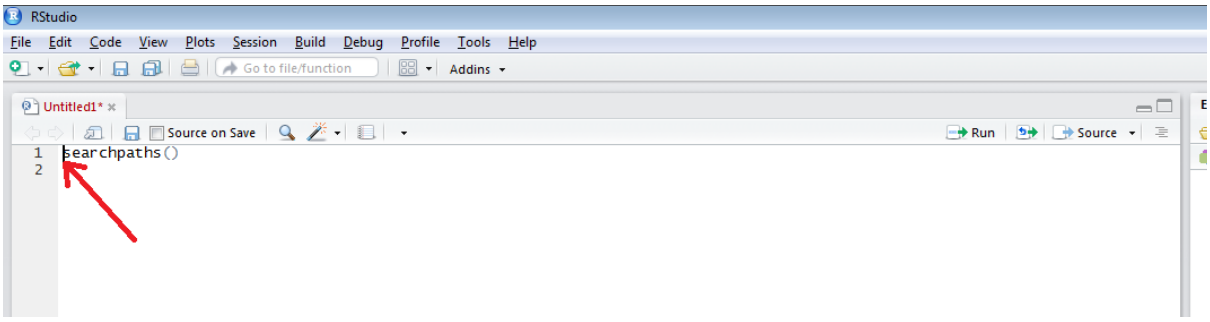
searchpaths()



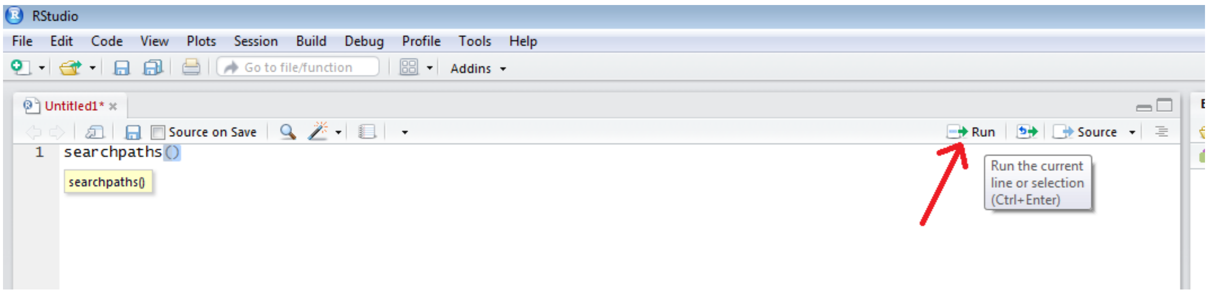


Intelisense in Rstudio will suggest the function as the keystrokes take place.

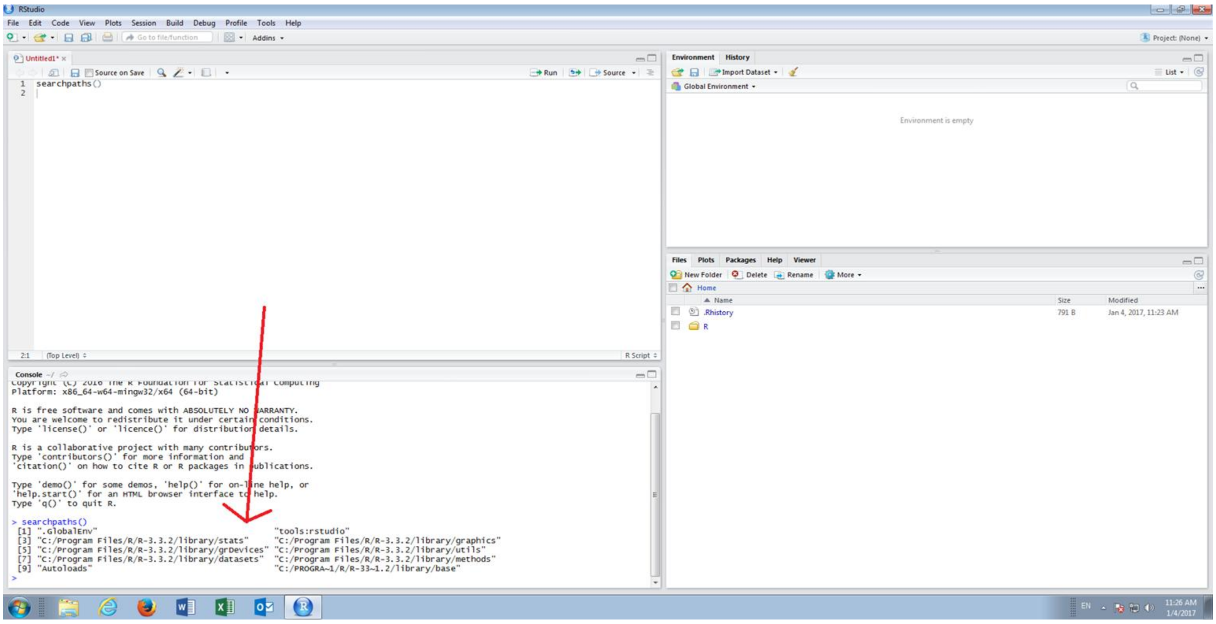
Upon the function having been written in the script editor, move the cursor to the start of the line (it will be implicitly understood that this has taken place in future procedures when the instrusction to Run to console is given):



Click the Run button or achive the same via a Ctrl+Enter key combination:



Runing sends the line of script to the console for execution:



Upon close inspection, it can be seen that the packages and their file location in the R execution directory have been listed in the console:

[1] ".GlobalEnv"   "tools:rstudio"

[3] "C:/Program Files/R/R-3.3.2/library/stats"   "C:/Program Files/R/R-3.3.2/library/graphics"

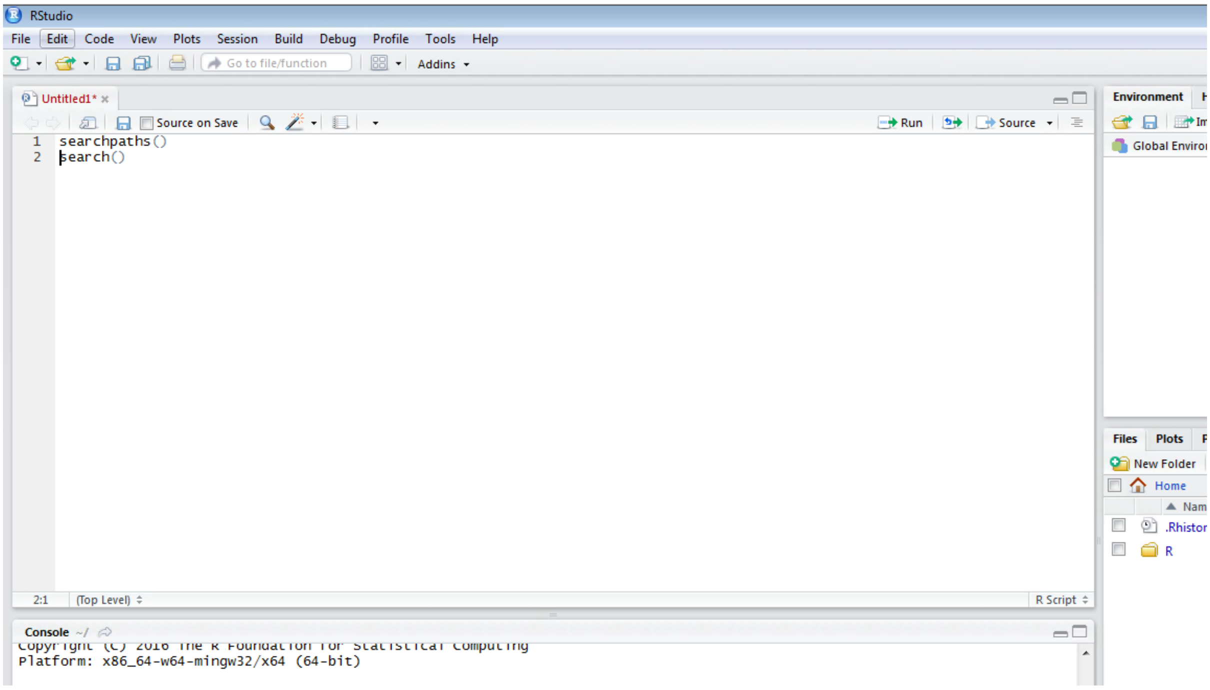
[5] "C:/Program Files/R/R-3.3.2/library/grDevices" "C:/Program Files/R/R-3.3.2/library/utils"

[7] "C:/Program Files/R/R-3.3.2/library/datasets" "C:/Program Files/R/R-3.3.2/library/methods"

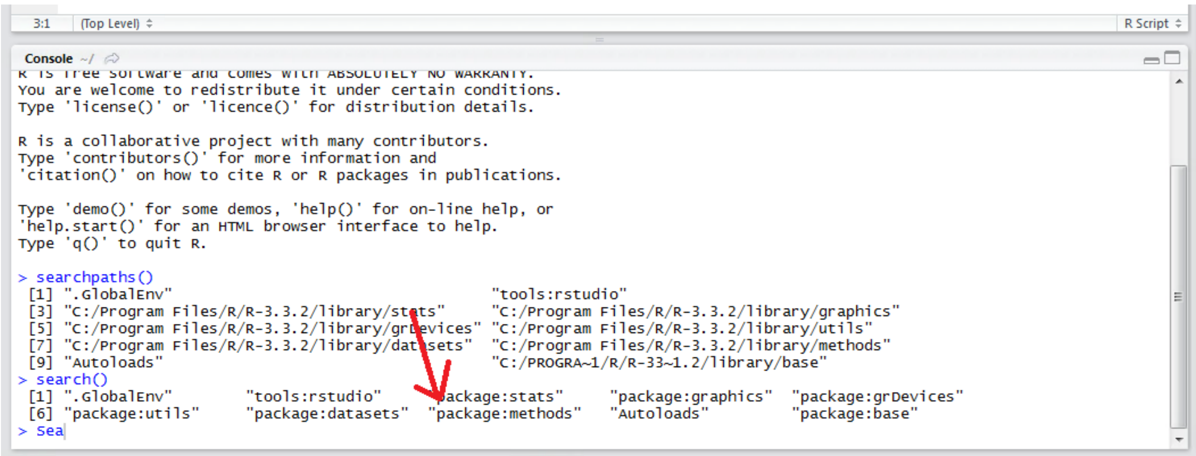
[9] "Autoloads"   "C:/PROGRA~1/R/R-33~1.2/library/base"

A more environment focussed presentation of the pakages can be achived by creating a new line in the script editor and typing:

search()



Run to console:

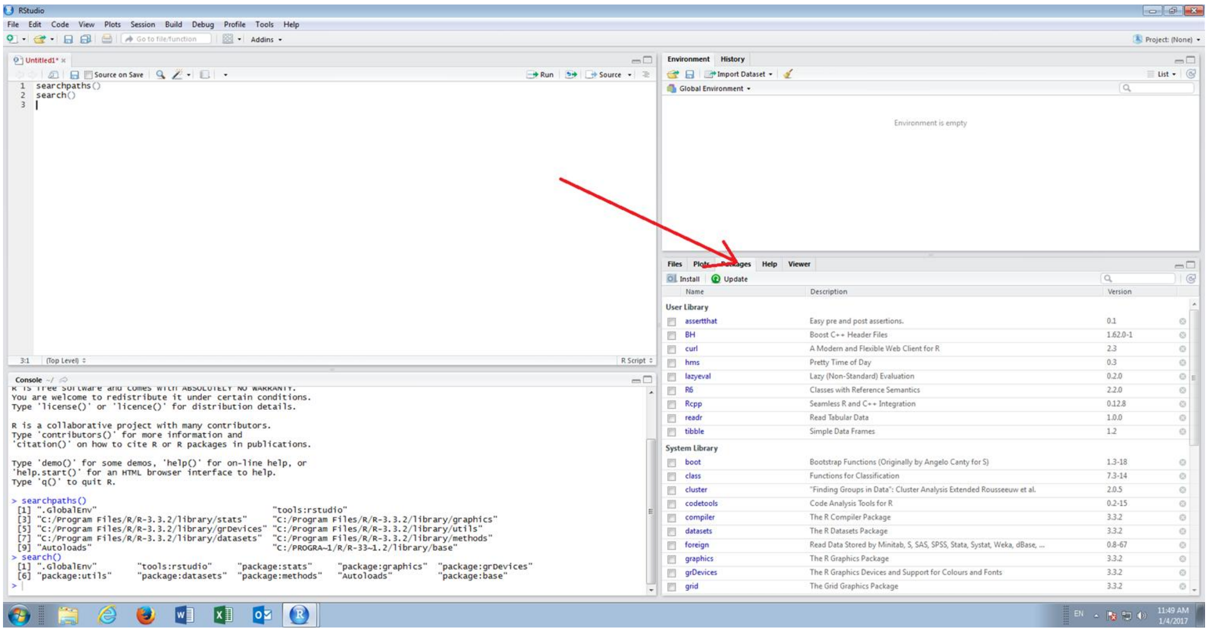


The search() function gives a more consise list of the packages that are available and loaded.

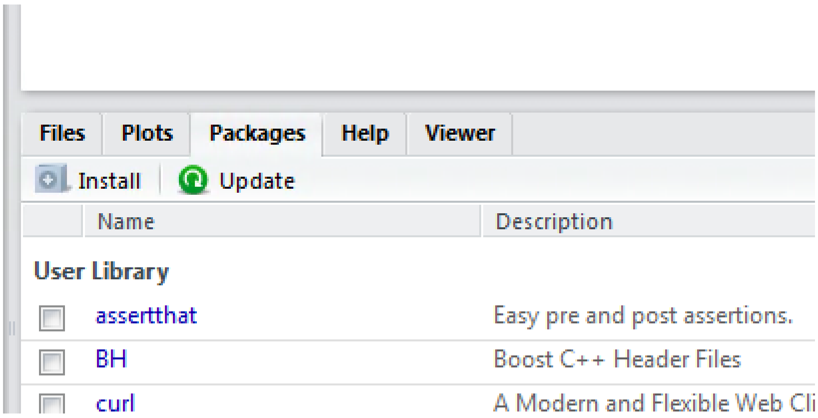
Procedure 7: Browsing and Installing Packages.

For the purposes of these procedures all external packages will be sourced from CRAN via Rstudio. In this procedure the graphics and plotting package titled ggplot2 will be installed.

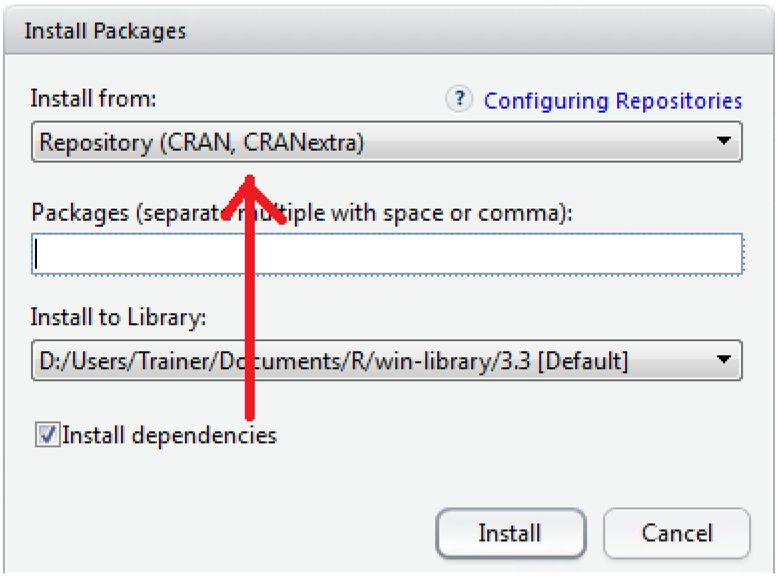
Navigate to the Packages pane, clicking the tab if nessecary, in the botton right hand corner of RStudio:



Click on the button Install:

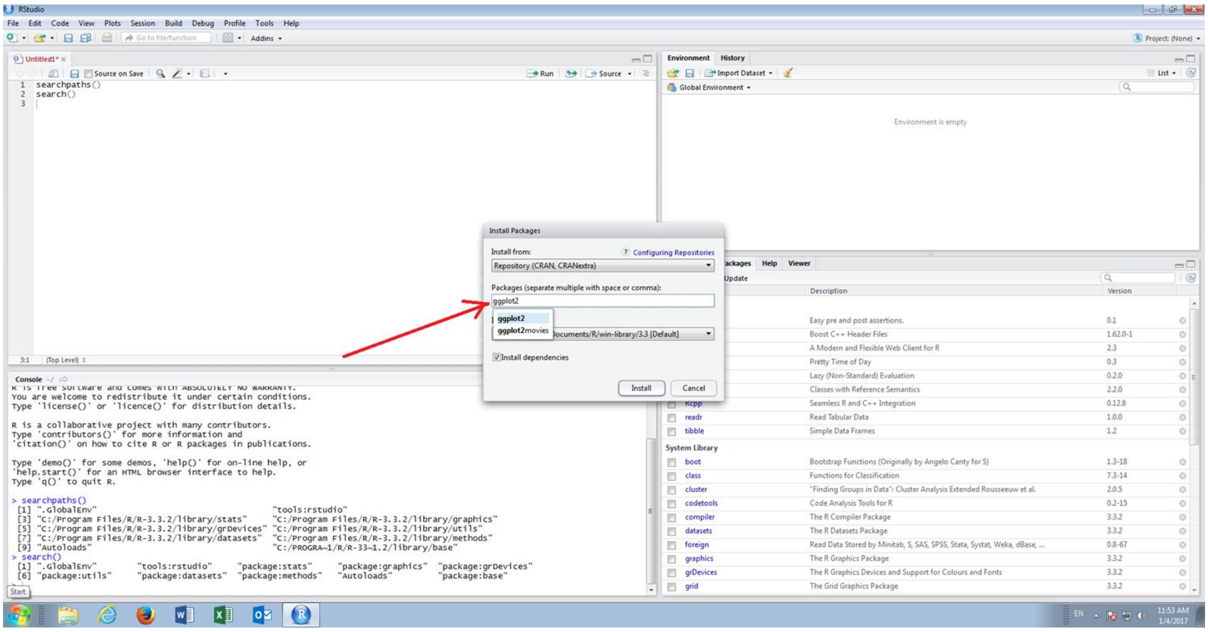


The Install Package dialog box will display, defaulting to the CRAN mirror:

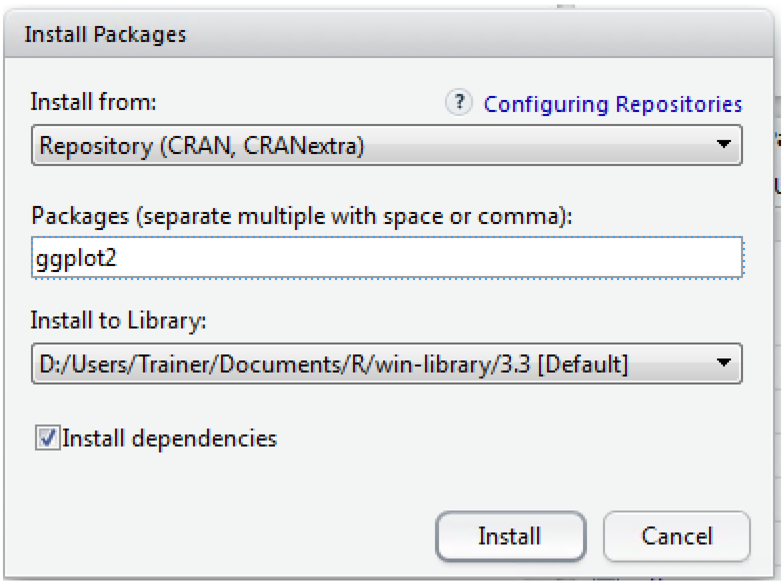


To search for a package by name, type the name in the package textbox:

ggplot2



Autocomplete will suggest two packages having reviewed potential matched on CRAN, accept \ click on the suggested ggplot2:



Always keep the Install Dependencies button as checked. Clicking install will send commands to the console to install the packages:

```
Console ~/
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/ggplot2_2.2.1.zip'
Content type 'application/zip' length 2760804 bytes (2.6 MB)
downloaded 2.6 MB

package ‘stringi’ successfully unpacked and MD5 sums checked
package ‘magrittr’ successfully unpacked and MD5 sums checked
package ‘colorspace’ successfully unpacked and MD5 sums checked
package ‘stringr’ successfully unpacked and MD5 sums checked
package ‘RColorBrewer’ successfully unpacked and MD5 sums checked
package ‘dichromat’ successfully unpacked and MD5 sums checked
package ‘munsell’ successfully unpacked and MD5 sums checked
package ‘labeling’ successfully unpacked and MD5 sums checked
package ‘digest’ successfully unpacked and MD5 sums checked
package ‘gtable’ successfully unpacked and MD5 sums checked
package ‘plyr’ successfully unpacked and MD5 sums checked
package ‘reshape2’ successfully unpacked and MD5 sums checked
package ‘scales’ successfully unpacked and MD5 sums checked
package ‘ggplot2’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpEzjxzZ\downloaded_packages
>
```

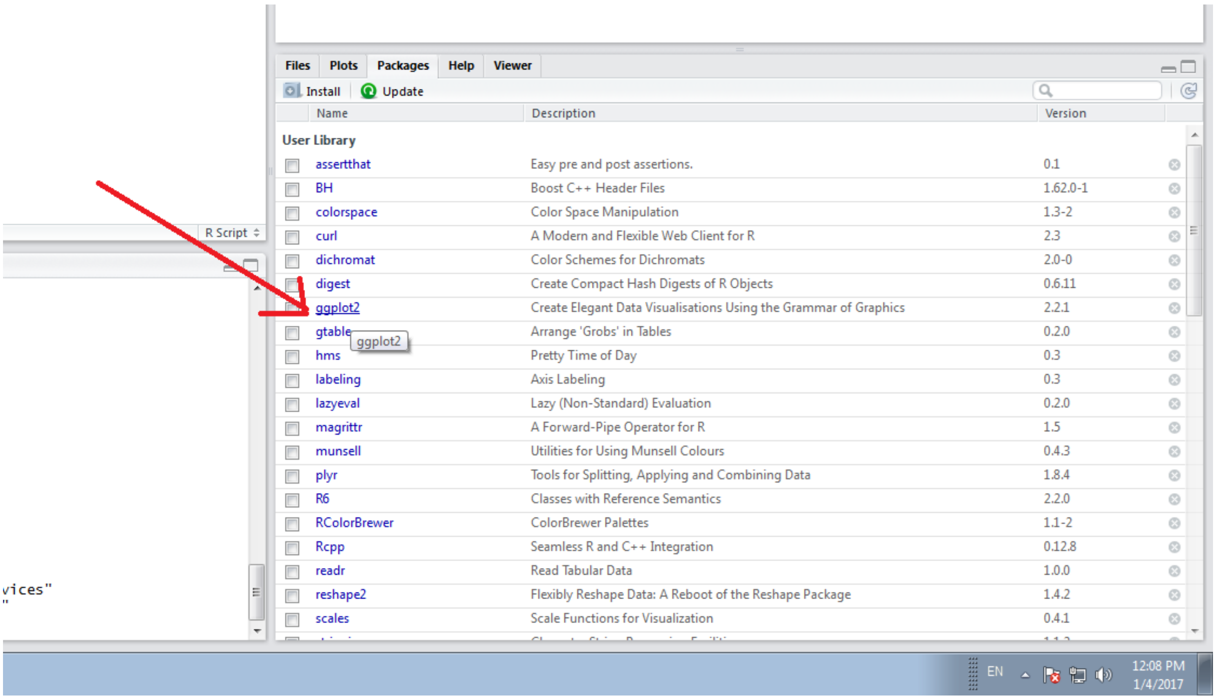
The package is now installed.

Executing the search() function it can be observed however that the package appears not to be loaded:

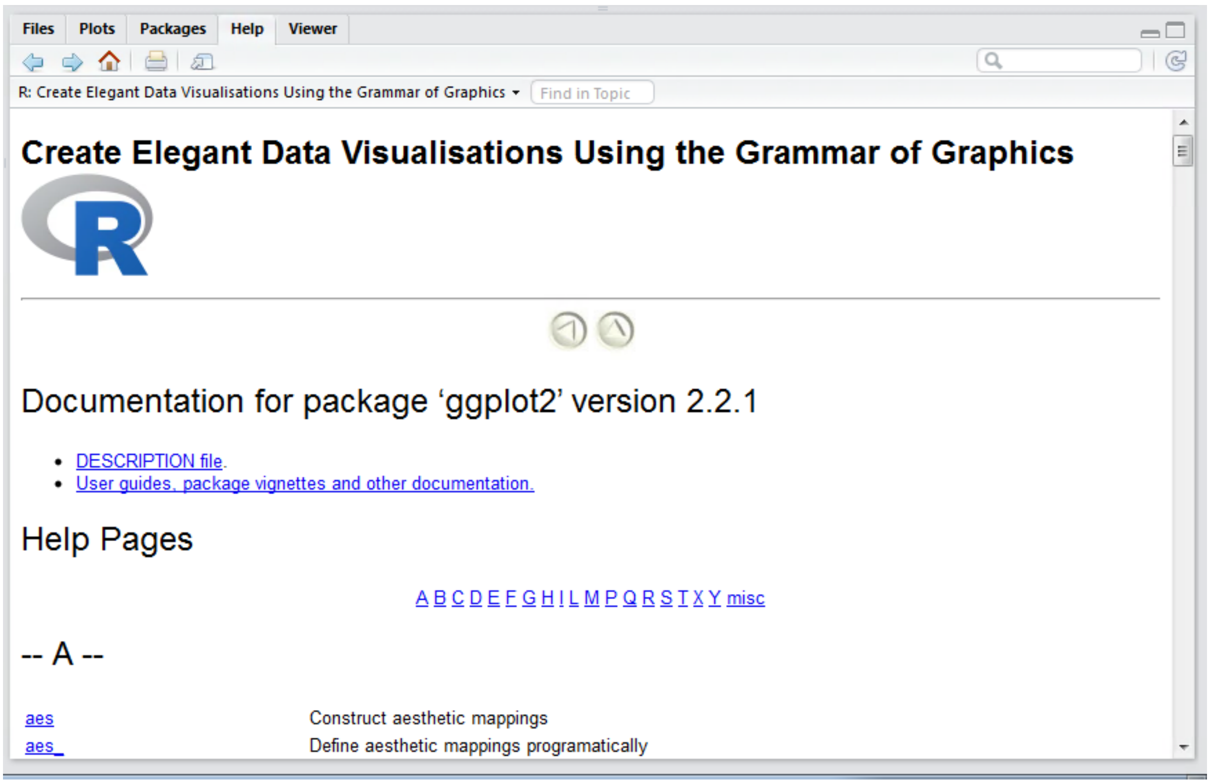
```
Console ~/
package ‘stringi’ successfully unpacked and MD5 sums checked
package ‘magrittr’ successfully unpacked and MD5 sums checked
package ‘colorspace’ successfully unpacked and MD5 sums checked
package ‘stringr’ successfully unpacked and MD5 sums checked
package ‘RColorBrewer’ successfully unpacked and MD5 sums checked
package ‘dichromat’ successfully unpacked and MD5 sums checked
package ‘munsell’ successfully unpacked and MD5 sums checked
package ‘labeling’ successfully unpacked and MD5 sums checked
package ‘digest’ successfully unpacked and MD5 sums checked
package ‘gtable’ successfully unpacked and MD5 sums checked
package ‘plyr’ successfully unpacked and MD5 sums checked
package ‘reshape2’ successfully unpacked and MD5 sums checked
package ‘scales’ successfully unpacked and MD5 sums checked
package ‘ggplot2’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpEzjxzZ\downloaded_packages
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"     "package:graphics"  "package:grDevices"
 [6] "package:utils"     "package:datasets"  "package:methods"   "Autoloads"         "package:base"
>
```

Procedure 8: Review Help and Documentation.

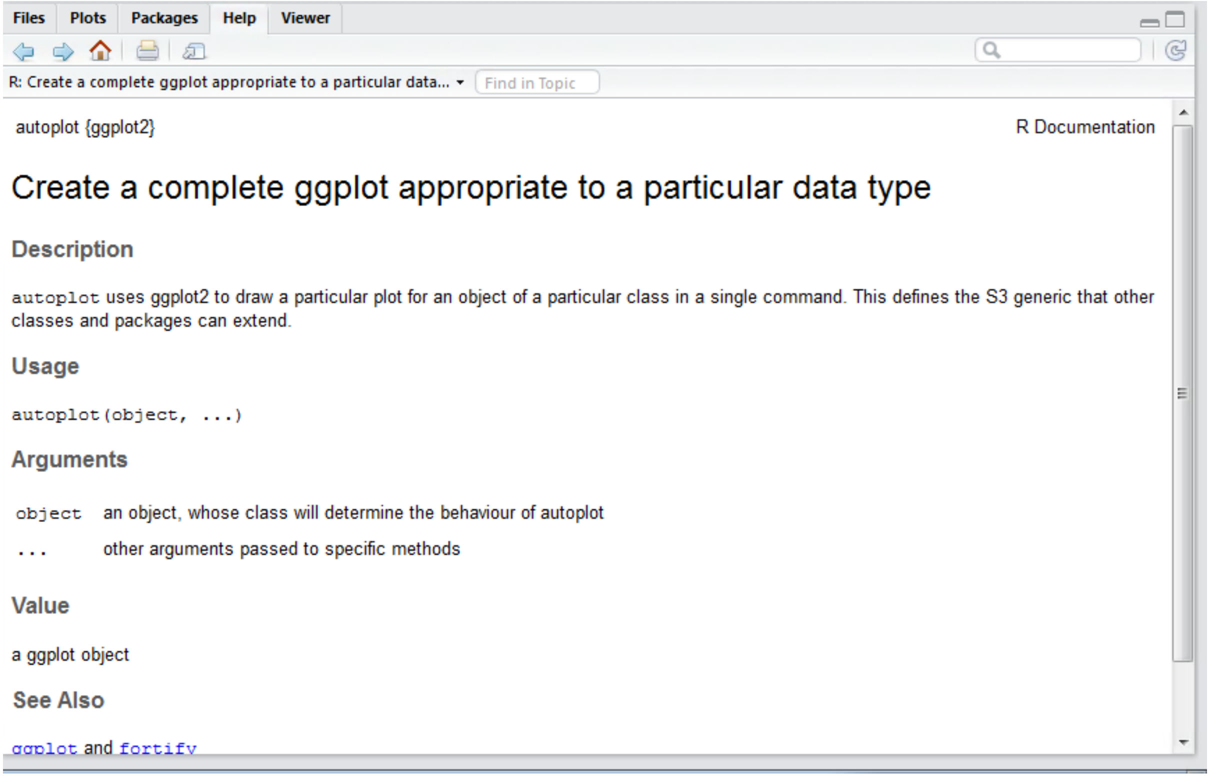
In the packages pane a list of all packages installed has been presented. It can be seen, following the execution of Procedure 7, that ggplot2 is now installed. It is customary for packages to carry good documentation, which can be accessed by clicking on the hyperlink overlaying the package name:



Clicking on the link immediate navigation to the packages documentation takes place:



This feature provides a more intuative means to navigate to documentation for functions. In this example, scroll down and clock on the function link autoplot:



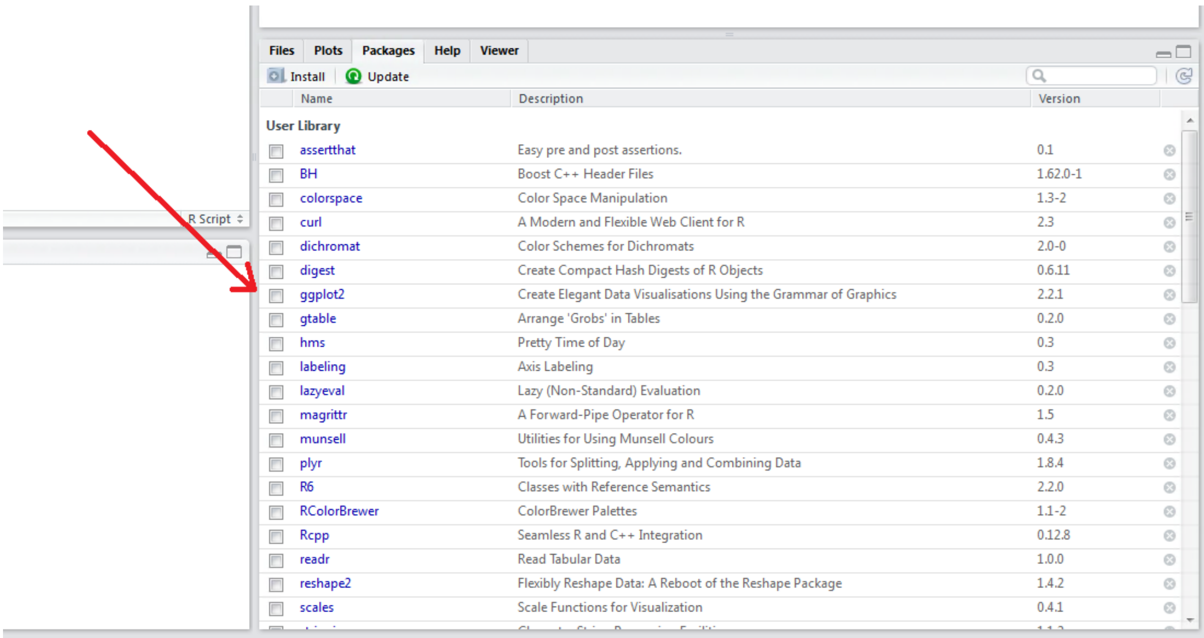
It can be seen that the documentation is displayed, navigating away from the index.

Procedure 9: Load and Unload Packages in RStudio.

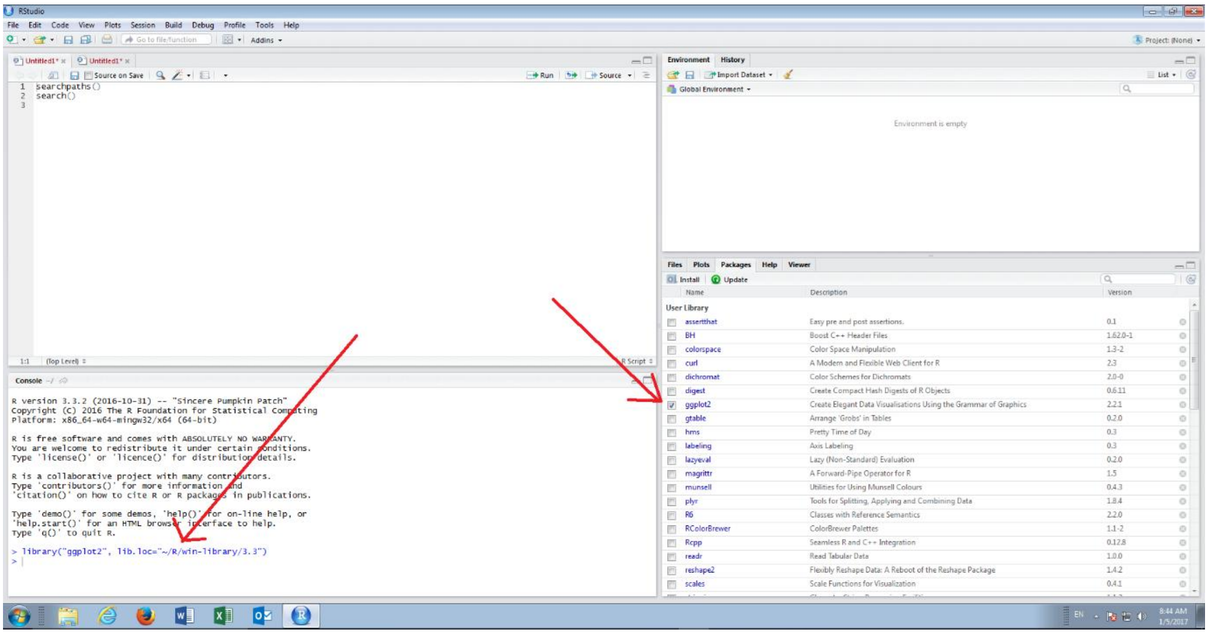
When a package is installed it is not by default available for use, to save memory and resources. Loading a package in RStudio is an extremly simple toggle process which will send the command to console to load a specific package on select, unload on deselect.

Loading a package uses the library() function, invoked before a script is run.

Navigate to the packages pane in the bottom right hand corner of the RStudio, clicking on the tab if nessecary.

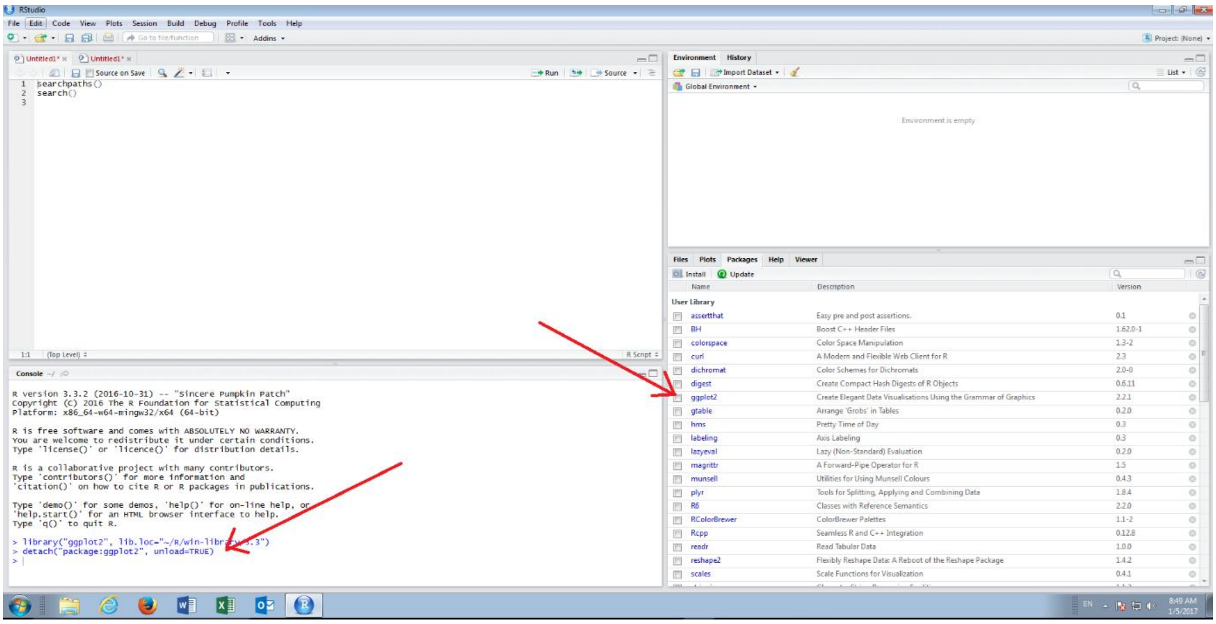


Note the package that was installed in procedure 7, ggplot2, and the check box to the left hand side of the package name. To load the package simply select the textbox via a click of the mouse:



On selection of the checkbox the library() fuction, complete with the required paramaters, will be processed in the console. It can also be observed that the location of the package has been specidfied in this function call, although that is not strictly nessecary.

To unload the package, deselect the checkbox in the packagages pane next to ggplot2:

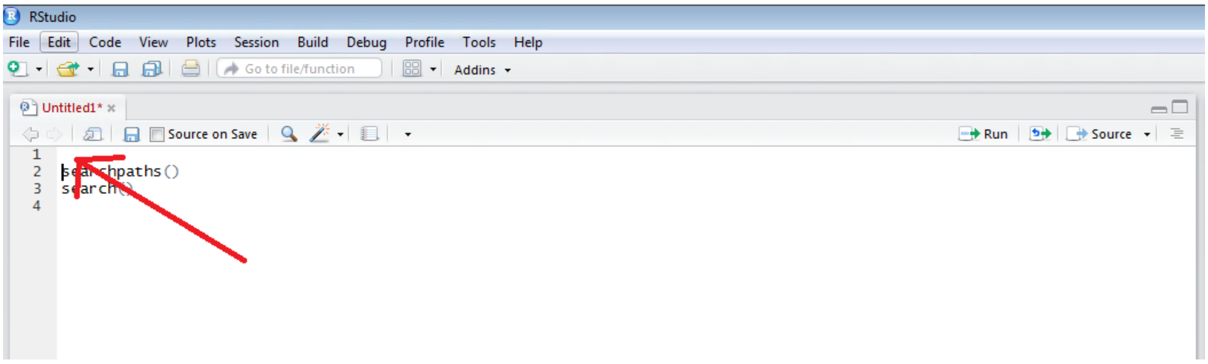


On deselection of the checkbox the detatch() function is sent to the console for the package (notice the string will match the return of the search() function).

Procedure 10: Load Packages using Script.

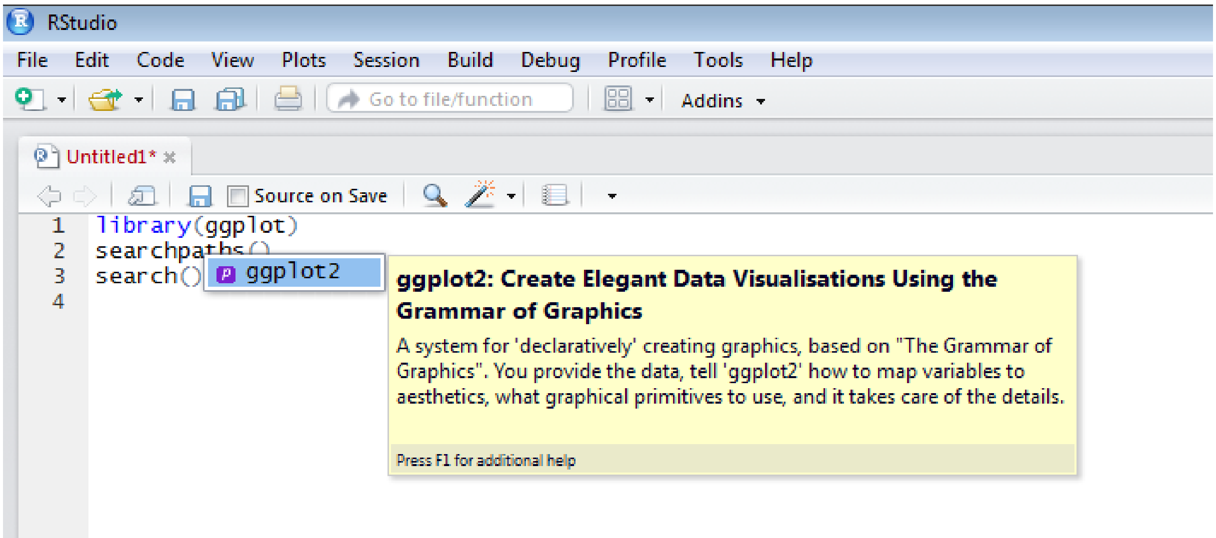
While the toggle function is a useful feature of RStudio, the intention is to maintain a script Active, Console Passive approach, henceforth it is important to ensure that the librarby() function call, to be streamed to the console is moved to the head of the script and that the detach() function is moved to the base of the script.

Start by navigating to the very top of the script and create a new line in the script editor. Navigate to the start of the first line and press the enter key:

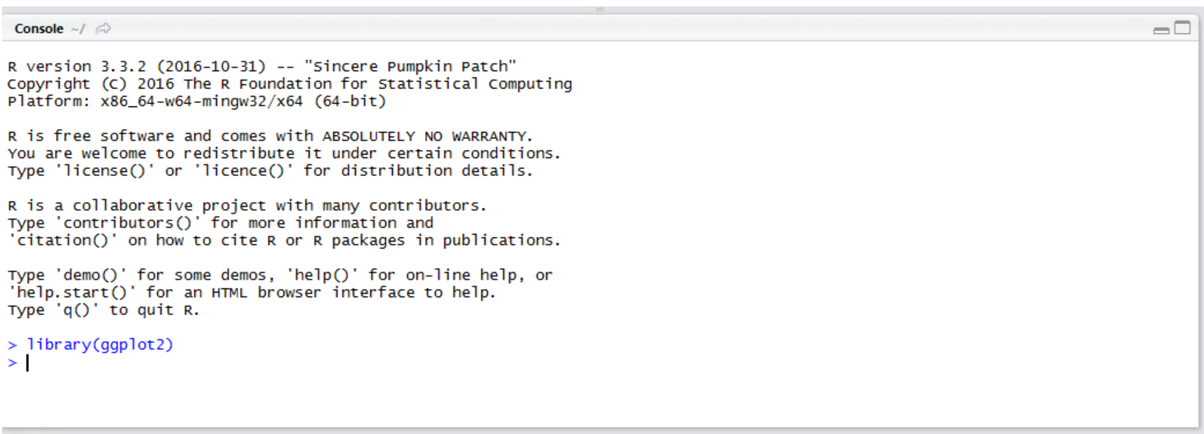


Invoking the library() function, type:

library(ggplot2)



Intelisense will look through the search path and suggest some packages, and this can also be autocompleted to ggplot2. Upon completion of the line, run the script line to the console:

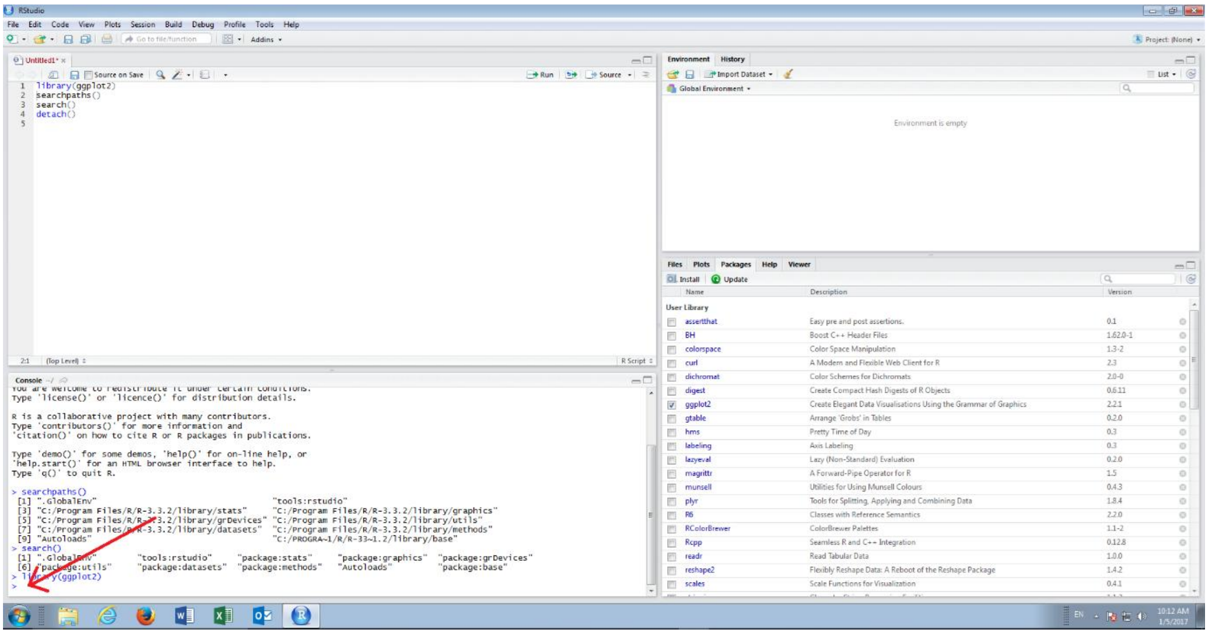


The ggplot library is now loaded as the first line of the script.

Procedure 11: List all Functions in a Package.

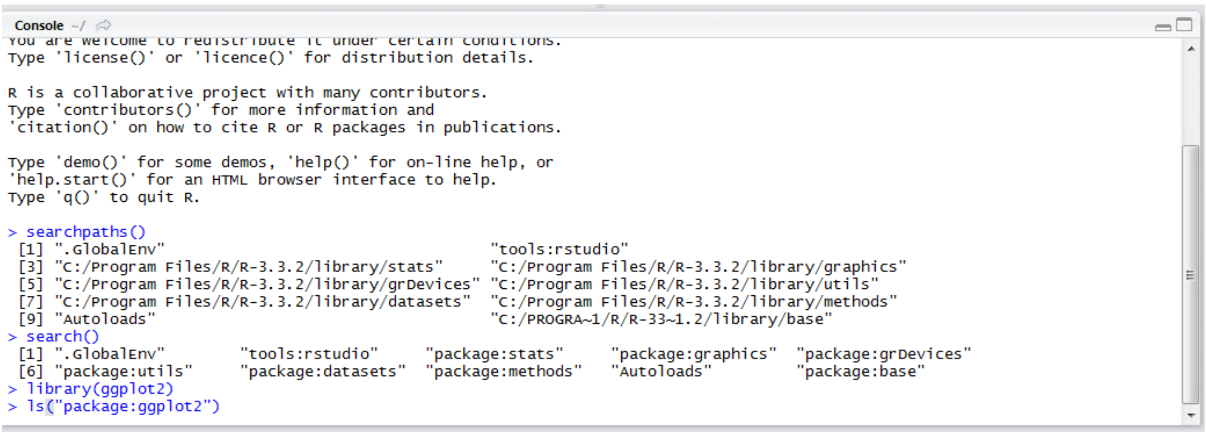
Once a package is loaded, beyond using the help as detailed in procedure 8, an understanding of all the functions available to the package can be obtained. Although a script active, console passive approach is advocated this procedure is one of the few occasions where it is more appropriate to use the console directly rather than clutter up the script.

Click on the console window in the bottom left hand corner of RStudio:



Type directly into the console:

ls("package:ggplot2")



Press the Enter key to execute the script:

```
Console ~/
[321] "ScaleDiscreteIdentity"  "ScaleDiscretePosition"  "seals"                 "sec_axis"
[325] "should_stop"            "Stat"                   "stat_bin"              "stat_bin_2d"
[329] "stat_bin_hex"           "stat_bin2d"             "stat_binhex"           "stat_boxplot"
[333] "stat_contour"           "stat_count"             "stat_density"          "stat_density_2d"
[337] "stat_density2d"         "stat_ecdf"              "stat_ellipse"          "stat_function"
[341] "stat_identity"          "stat_qq"                "stat_quantile"         "stat_smooth"
[345] "stat_spoke"             "stat_sum"               "stat_summary"          "stat_summary_2d"
[349] "stat_summary_bin"       "stat_summary_hex"       "stat_summary2d"        "stat_unique"
[353] "stat_ydensity"          "StatBin"                "StatBin2d"             "StatBindot"
[357] "StatBinhex"             "StatBoxplot"            "StatContour"           "StatCount"
[361] "StatDensity"            "StatDensity2d"          "StatEcdf"              "StatEllipse"
[365] "StatFunction"           "StatIdentity"           "StatQq"                "StatQuantile"
[369] "StatSmooth"             "StatSum"                "StatSummary"           "StatSummary2d"
[373] "StatSummaryBin"         "StatSummaryHex"         "StatUnique"            "StatYdensity"
[377] "theme"                  "theme_bw"               "theme_classic"         "theme_dark"
[381] "theme_get"              "theme_gray"             "theme_grey"            "theme_light"
[385] "theme_linedraw"         "theme_minimal"          "theme_replace"         "theme_set"
[389] "theme_update"           "theme_void"             "transform_position"    "txhousing"
[393] "unit"                   "update_geom_defaults"   "update_labels"         "update_stat_defaults"
[397] "waiver"                 "wrap_dims"              "xlab"                  "xlim"
[401] "ylab"                   "ylim"                   "zeroGrob"
>
```

A list of all functions in the package is returned.

Procedure 12: Use the help() function to explain a function.

If using RStudio, navigating to the documentation via the help pane is by far the easiest and most intuative means to access help. Taking the output of functions recalled in procedure 11, navigation to help can be triggered by invoking the help function.

As with procedure 11, this procedure is one of the few occasions where it is more approproate to target the console rather than the script.

To navigate to help, click on the conole input cursor:

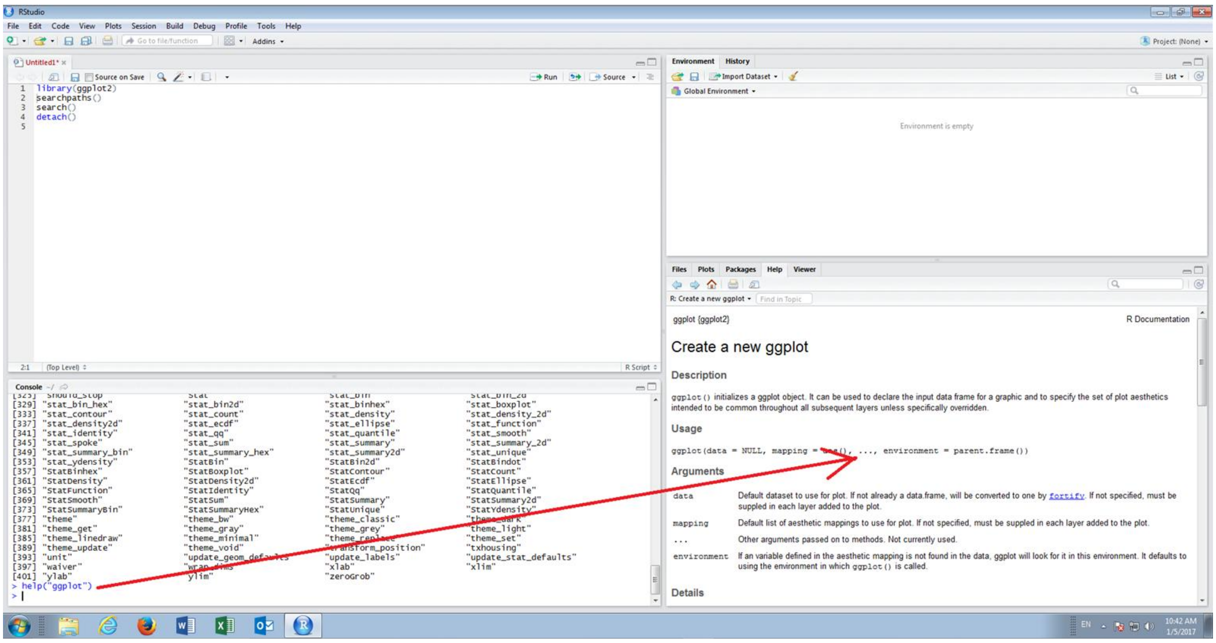

```
Console ~/
[321] "ScaleDiscreteIdentity"  "ScaleDiscretePosition"  "seals"                 "sec_axis"
[325] "should_stop"            "Stat"                   "stat_bin"              "stat_bin_2d"
[329] "stat_bin_hex"           "stat_bin2d"             "stat_binhex"           "stat_boxplot"
[333] "stat_contour"           "stat_count"             "stat_density"          "stat_density_2d"
[337] "stat_density2d"         "stat_ecdf"              "stat_ellipse"          "stat_function"
[341] "stat_identity"          "stat_qq"                "stat_quantile"         "stat_smooth"
[345] "stat_spoke"             "stat_sum"               "stat_summary"          "stat_summary_2d"
[349] "stat_summary_bin"       "stat_summary_hex"       "stat_summary2d"        "stat_unique"
[353] "stat_ydensity"          "StatBin"                "StatBin2d"             "StatBindot"
[357] "StatBinhex"             "StatBoxplot"            "StatContour"           "StatCount"
[361] "StatDensity"            "StatDensity2d"          "StatEcdf"              "StatEllipse"
[365] "StatFunction"           "StatIdentity"           "StatQq"                "StatQuantile"
[369] "StatSmooth"             "StatSum"                "StatSummary"           "StatSummary2d"
[373] "StatSummaryBin"         "StatSummaryHex"         "StatUnique"            "StatYdensity"
[377] "theme"                  "theme_bw"               "theme_classic"         "theme_dark"
[381] "theme_get"              "theme_gray"             "theme_grey"            "theme_light"
[385] "theme_linedraw"         "theme_minimal"          "theme_replace"         "theme_set"
[389] "theme_update"           "theme_void"             "transform_position"    "txhousing"
[393] "unit"                   "update_geom_defaults"   "update_labels"         "update_stat_defaults"
[397] "waiver"                 "wrap_dims"              "xlab"                  "xlim"
[401] "ylab"                   "ylim"                   "zeroGrob"
>
```


Type:

help("ggplot")

```
Console ~/
[321] "ScaleDiscreteIdentity"  "ScaleDiscretePosition"  "seals"                 "sec_axis"
[325] "should_stop"            "Stat"                   "stat_bin"              "stat_bin_2d"
[329] "stat_bin_hex"           "stat_bin2d"             "stat_binhex"           "stat_boxplot"
[333] "stat_contour"           "stat_count"             "stat_density"          "stat_density_2d"
[337] "stat_density2d"         "stat_ecdf"              "stat_ellipse"          "stat_function"
[341] "stat_identity"          "stat_qq"                "stat_quantile"         "stat_smooth"
[345] "stat_spoke"             "stat_sum"               "stat_summary"          "stat_summary_2d"
[349] "stat_summary_bin"       "stat_summary_hex"       "stat_summary2d"        "stat_unique"
[353] "stat_ydensity"          "StatBin"                "StatBin2d"             "StatBindot"
[357] "StatBinhex"             "StatBoxplot"            "StatContour"           "StatCount"
[361] "StatDensity"            "StatDensity2d"          "StatEcdf"              "StatEllipse"
[365] "StatFunction"           "StatIdentity"           "StatQq"                "StatQuantile"
[369] "StatSmooth"             "StatSum"                "StatSummary"           "StatSummary2d"
[373] "StatSummaryBin"         "StatSummaryHex"         "StatUnique"            "StatYdensity"
[377] "theme"                  "theme_bw"               "theme_classic"         "theme_dark"
[381] "theme_get"              "theme_gray"             "theme_grey"            "theme_light"
[385] "theme_linedraw"         "theme_minimal"          "theme_replace"         "theme_set"
[389] "theme_update"           "theme_void"             "transform_position"    "txhousing"
[393] "unit"                   "update_geom_defaults"   "update_labels"         "update_stat_defaults"
help(topic, package = NULL, lib.loc = NULL, verbose = getOption("verbose"), try.all.packages = getOption("help.try.all.packages"), help_type = getOption("help_type"))
[401] "ylab"                   "ylim"                   "zeroGrob"
> help("ggplot")
```

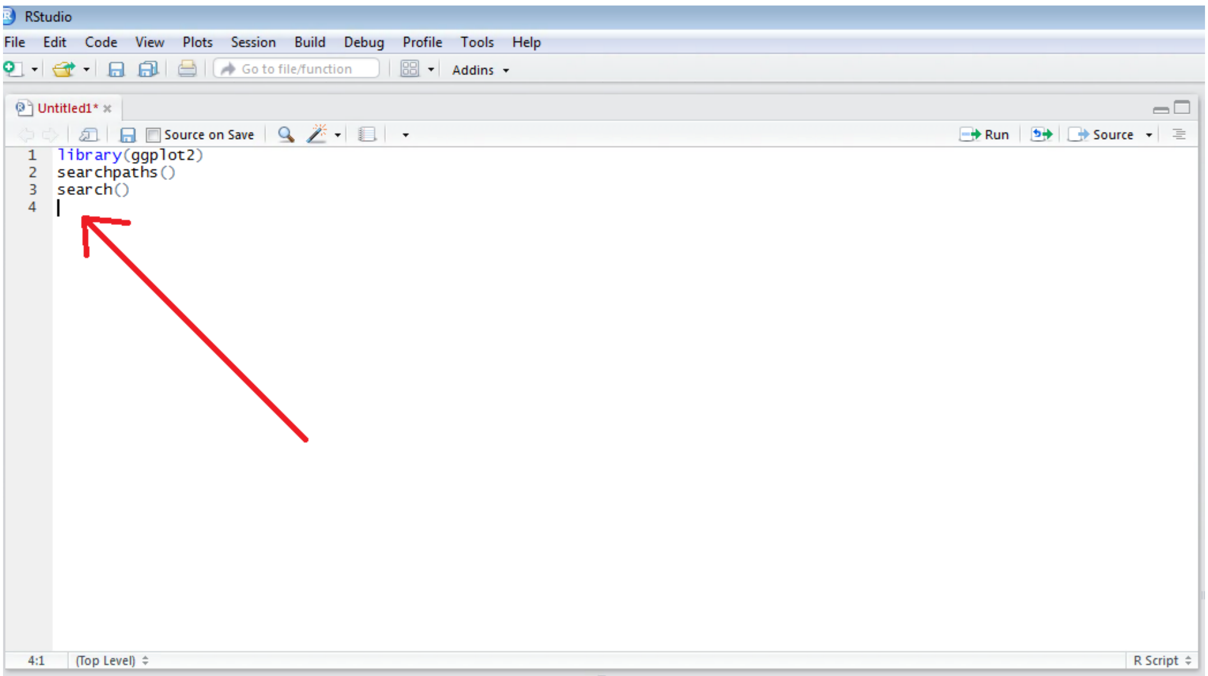
Press the Enter key to execute the line of script:



While operating in RStudio, the help will be displayed in the dedicated help pane. If operating in the console, the experience would be that the same text is written out to the console in text only. It follows that this procedure exists for the purposes of making help and documentation available universally in R.

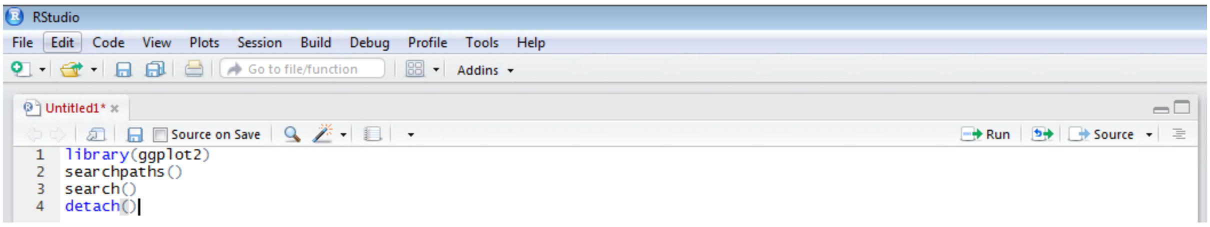
Procedure 13: Unloading a Package.

Navigate to the end of the script and create a new line:

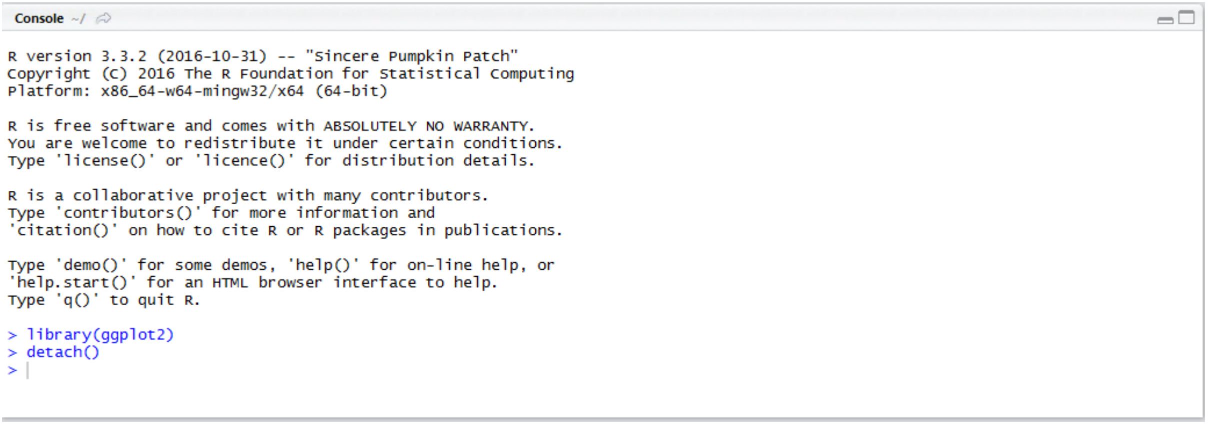


Type the detach() function as:

detach("package:ggplot2", unload = TRUE)



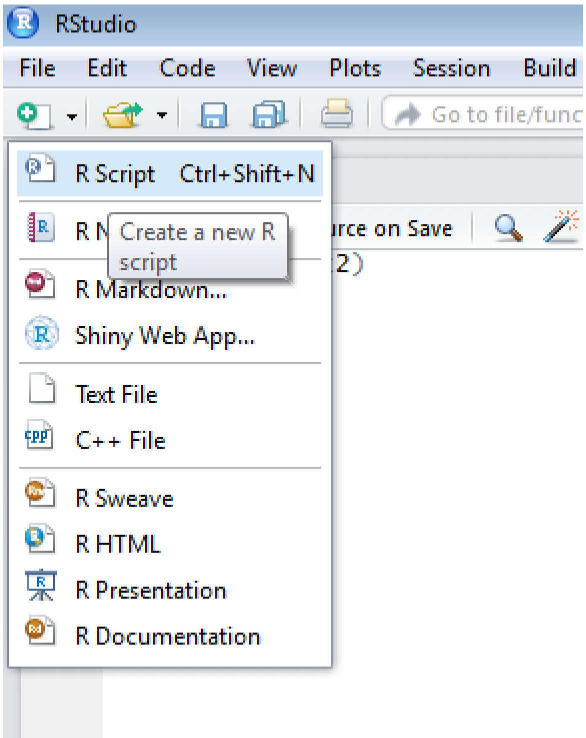
Run the line of script to the console:



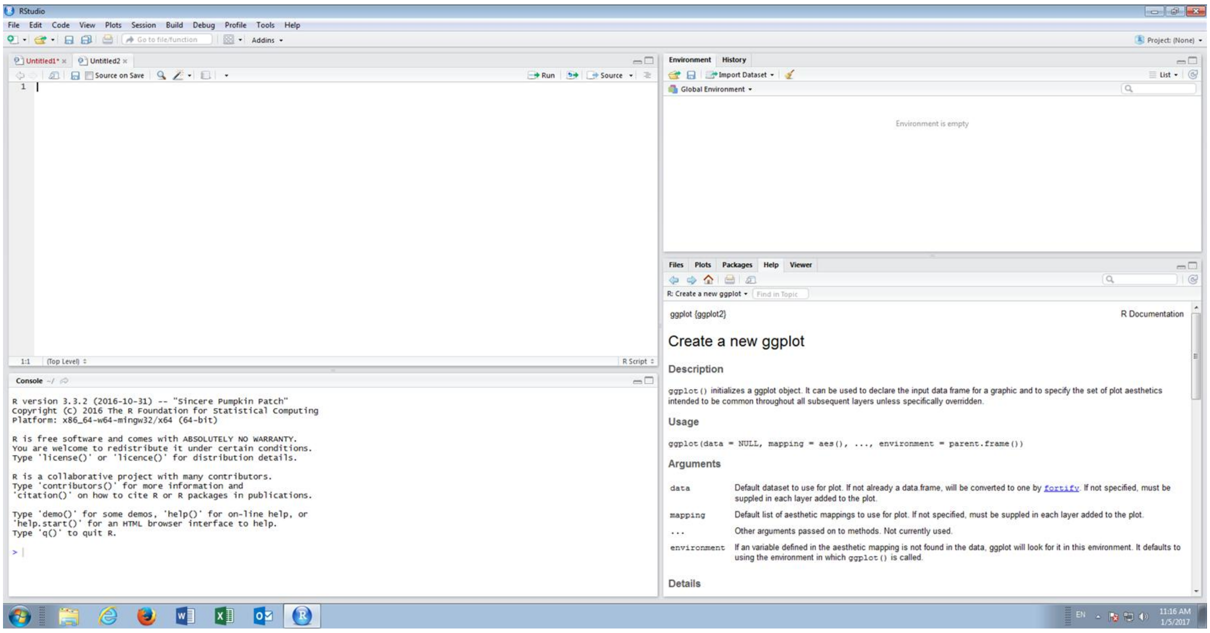
The package has now been removed from the R session and the scrpt is essentially, tidying itself up.

Procedure 14: Creating a Numeric Variable by Assignment.

Start by creating a new script as procedure 5:

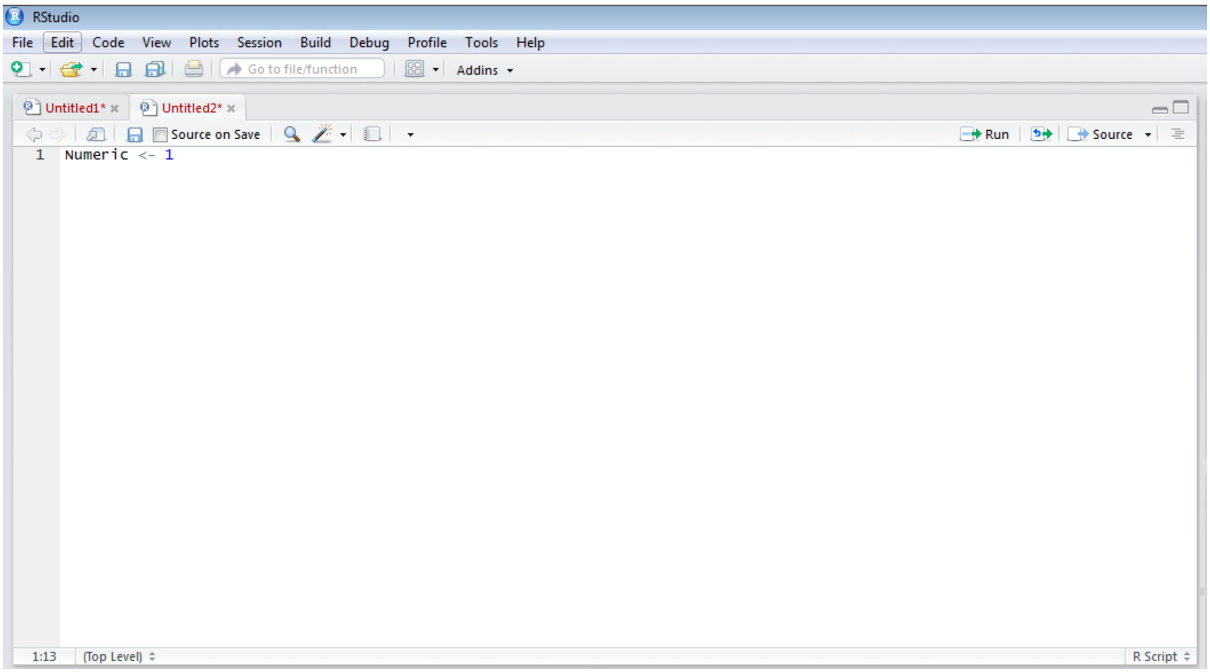


A blank script window will be created that will be the target:

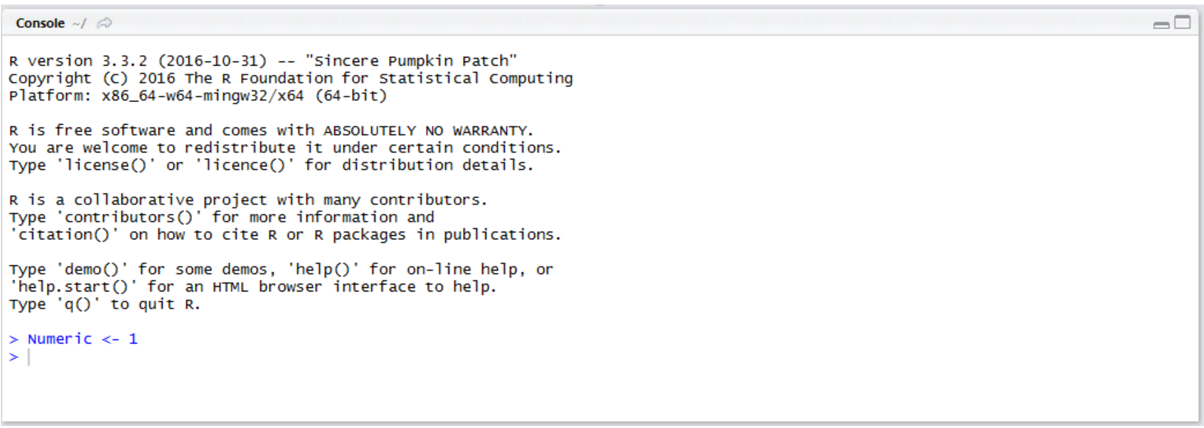


Variables in R are created by assignment, the process of setting a value. The operator or command for assignment is the chracter combination of "<-". To create and assign a numeric variable start by typing into the script window:

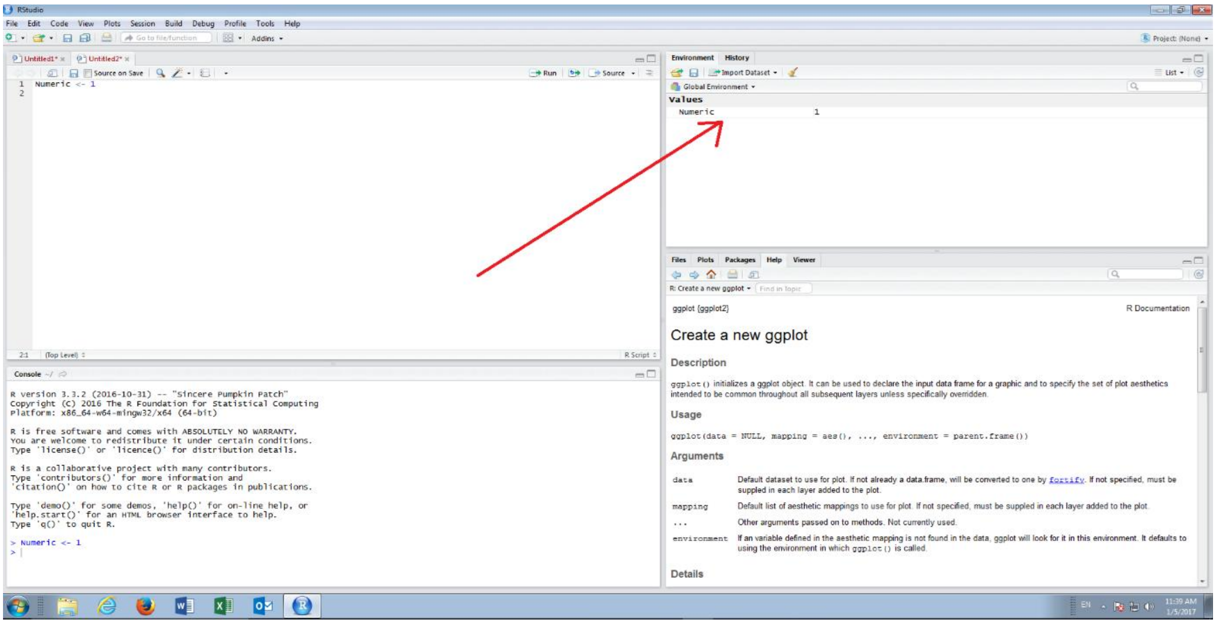
Numeric <- 1



Run the script to console:

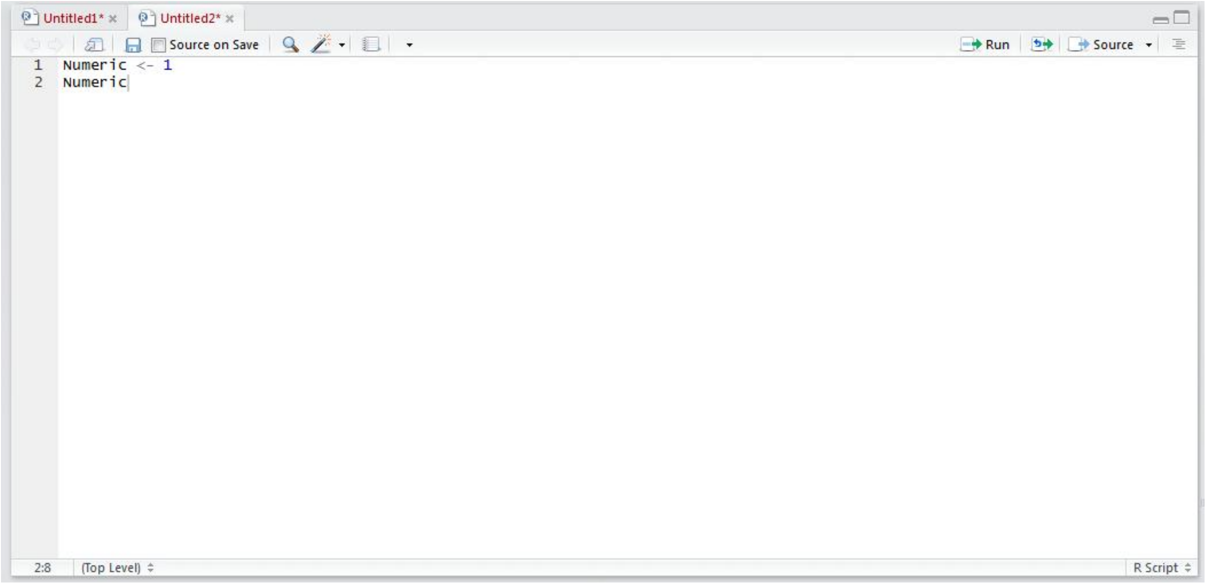


A variable with the name Numeric has now been created. It can be seen that RStudio has also recognised the creation of a new variable in the Environment Values pane towards the top right hand side of RStudio:

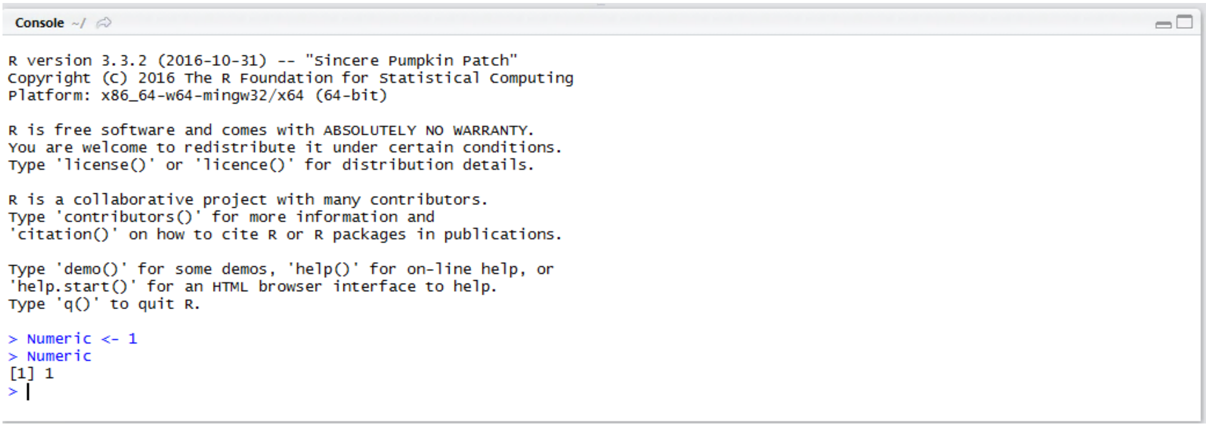


The variable can also be referenced in the script by simply typing the variable name and runing the script to console. In this example, create a new line in the script and type the name of the variable:

Numeric



Run the line of script to console:

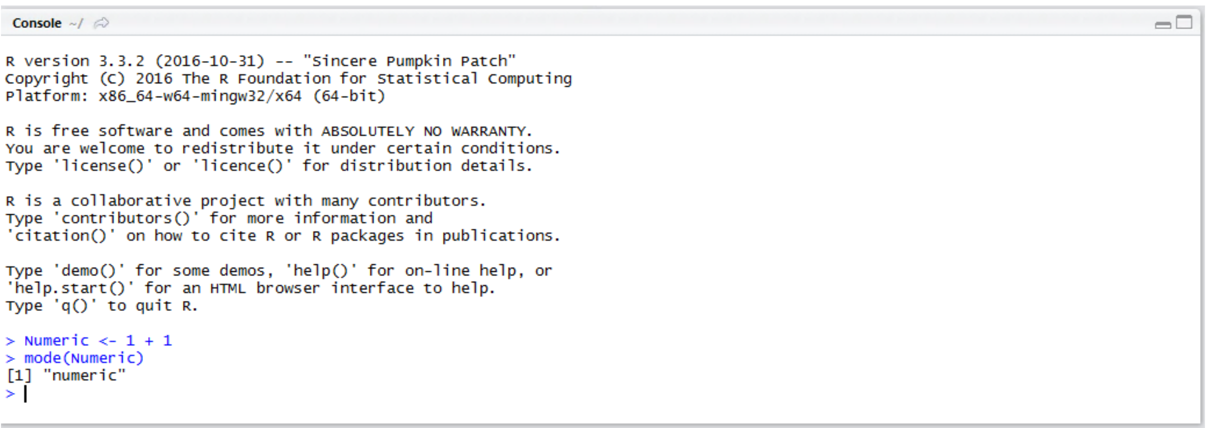


It can be seen that the assigned value is written out.

The mode() function is intended to disclose the variable type, taking the variable name as the paramater. Create a new line in the script editor and disclose the variable type, start by typing:

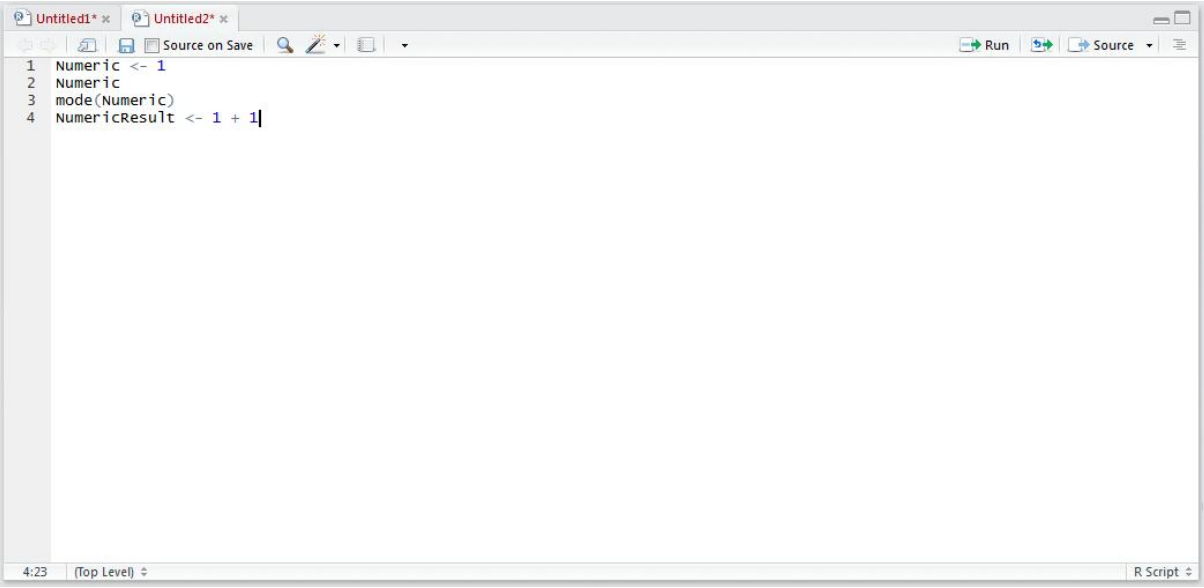
mode(Numeric)

Run the line of script to console:

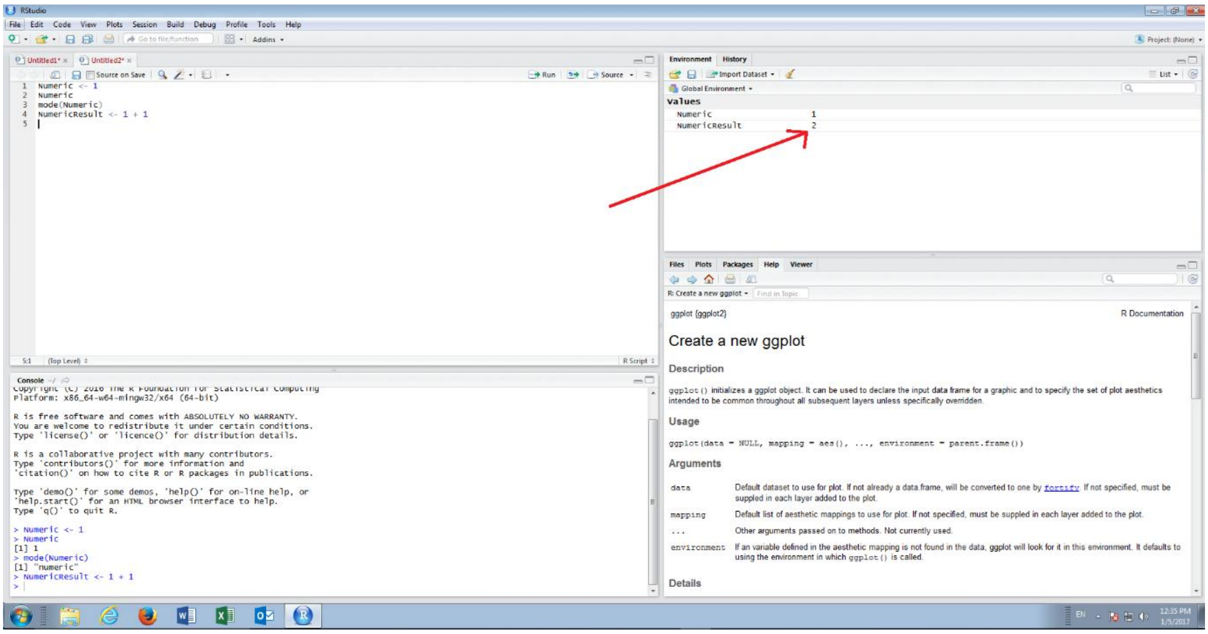


It can be observed that the variable type has been returned as numeric. It is also possible to assign a variable as the result of aritimetic or function output. For example, type into the script editor:

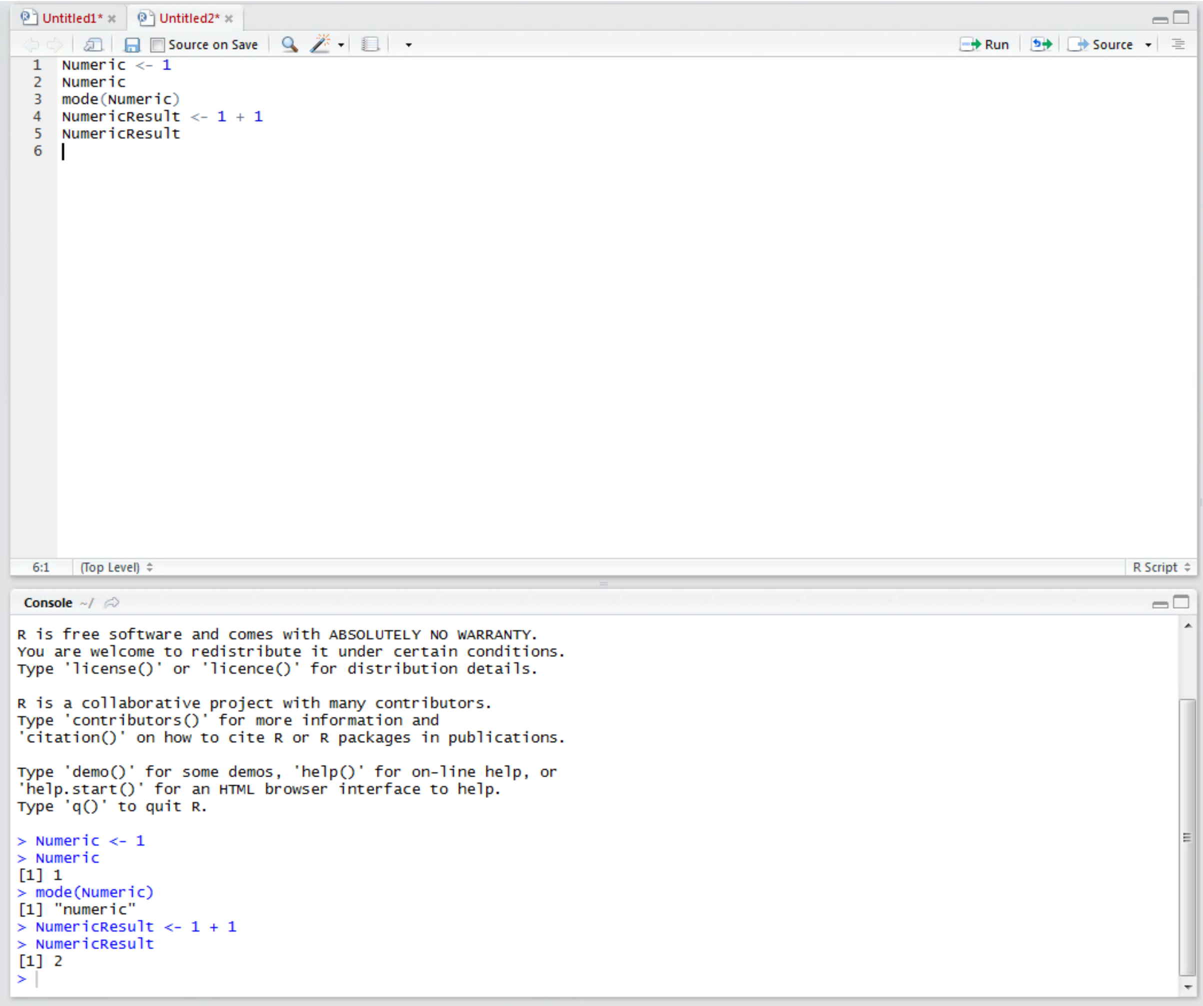
NumericResult <- 1 + 1



Run the line of script to the console:



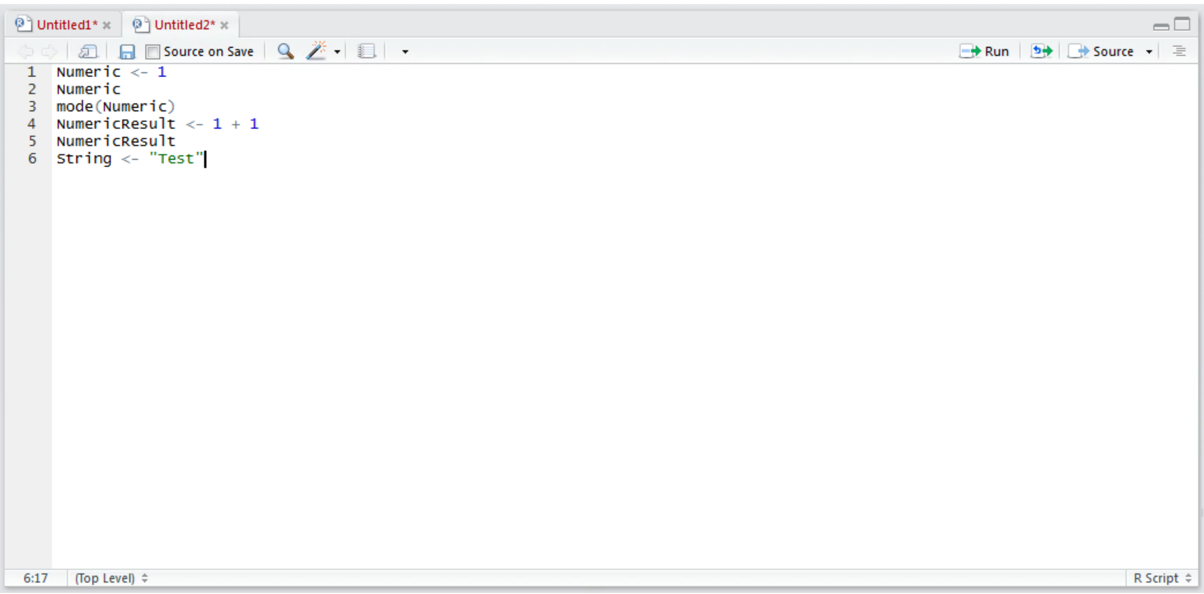
It can be observed that the NumericVariable has been created and is available in the Environment Variables windows, and it would also return in the console when referencing the variable directly:



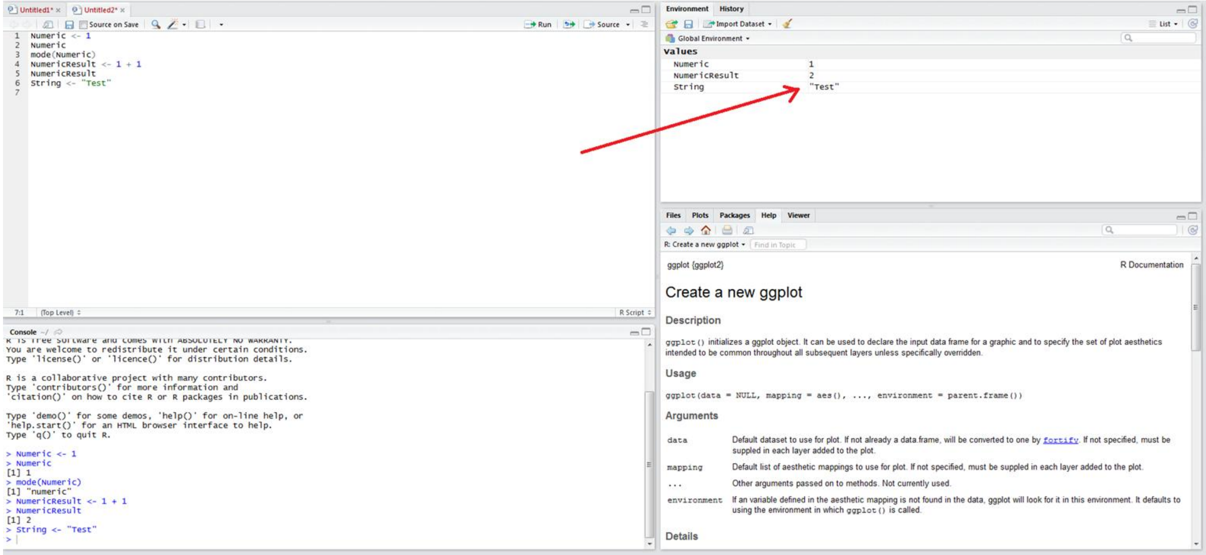
Procedure 15: Create a string variable by assignment.

Strings in R are surrounded by double quotation marks yet the assignment procedure is the same as numeric assignment. Start by creating a new line in the script editor and typing:

Char <- "Test"

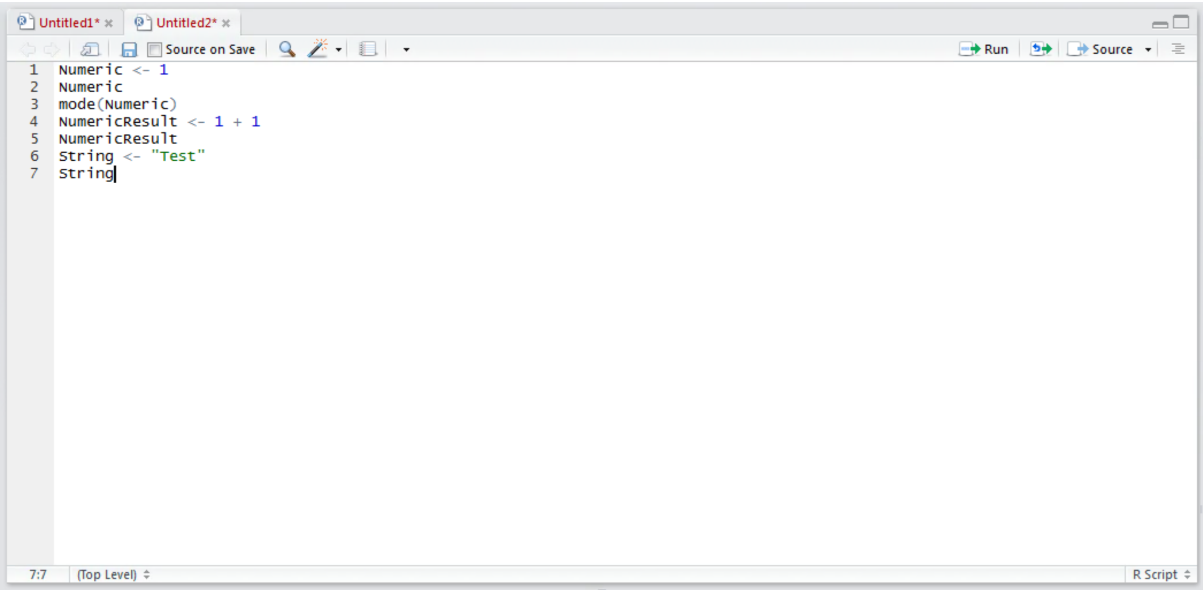


Run the script to console:



The new String value is written to the Environment pane. The variable is addressible from the script by typing the variable:

String



Run the line of script to console:

```
Console ~/
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Numeric <- 1
> Numeric
[1] 1
> mode(Numeric)
[1] "numeric"
> NumericResult <- 1 + 1
> NumericResult
[1] 2
> String <- "Test"
> String
[1] "Test"
>
```

Validate the variable type by using the mode() function. Type into the script pane:

mode(String)

```
Numeric <- 1
Numeric
mode(Numeric)
NumericResult <- 1 + 1
NumericResult
String <- "Test"
String
mode(String)
```

Run the line of script to console:

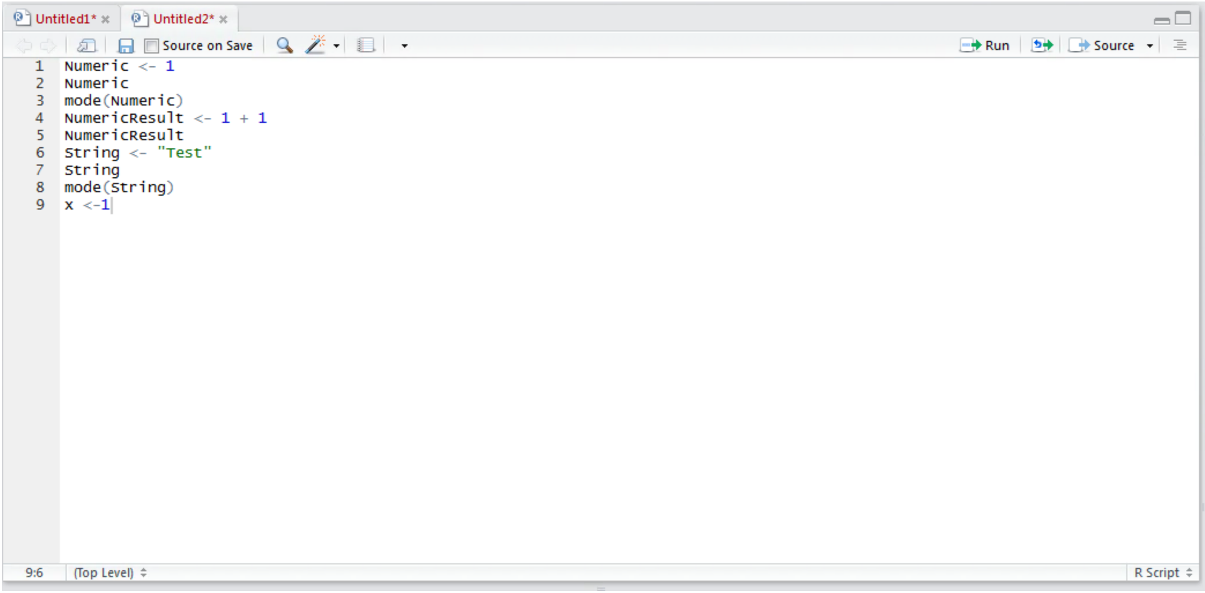
```
Console ~/
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Numeric <- 1
> Numeric
[1] 1
> mode(Numeric)
[1] "numeric"
> NumericResult <- 1 + 1
> NumericResult
[1] 2
> String <- "Test"
> String
[1] "Test"
> mode(String)
[1] "character"
>
```

It can be observed that the data type was defined as chracter upon assignment.

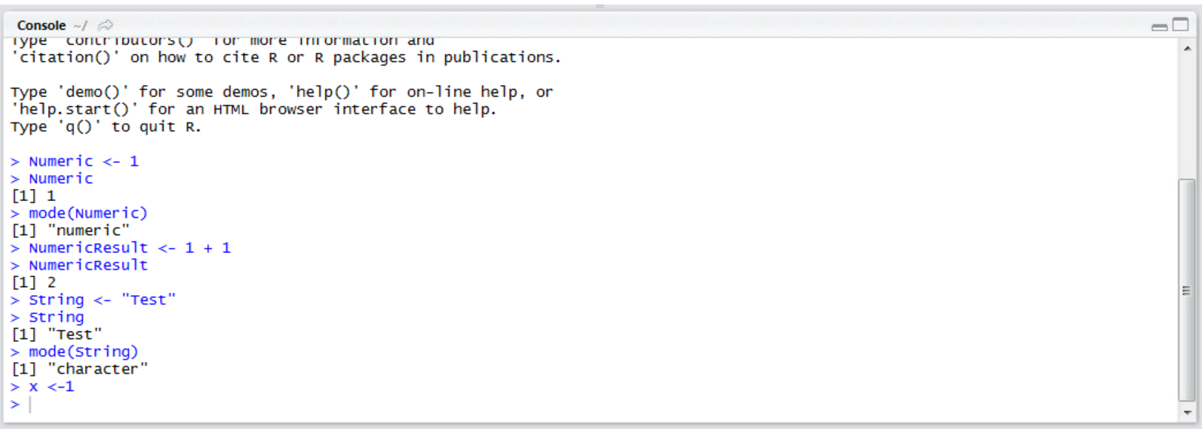
Procedure 16: Create a logical variable by assignment.

Logical variables are True or False values which are derived by logical assignment. To create a logical variable as the result of an evaluation assignment, start by creating a variable x by typing:

x <- 1

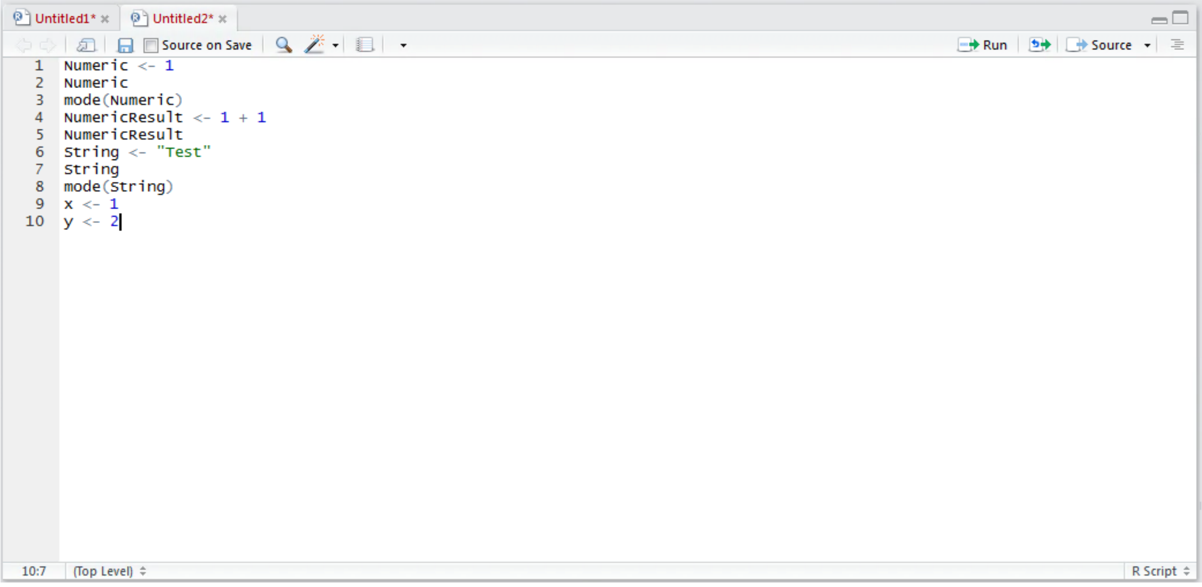


Run the line of script to console:

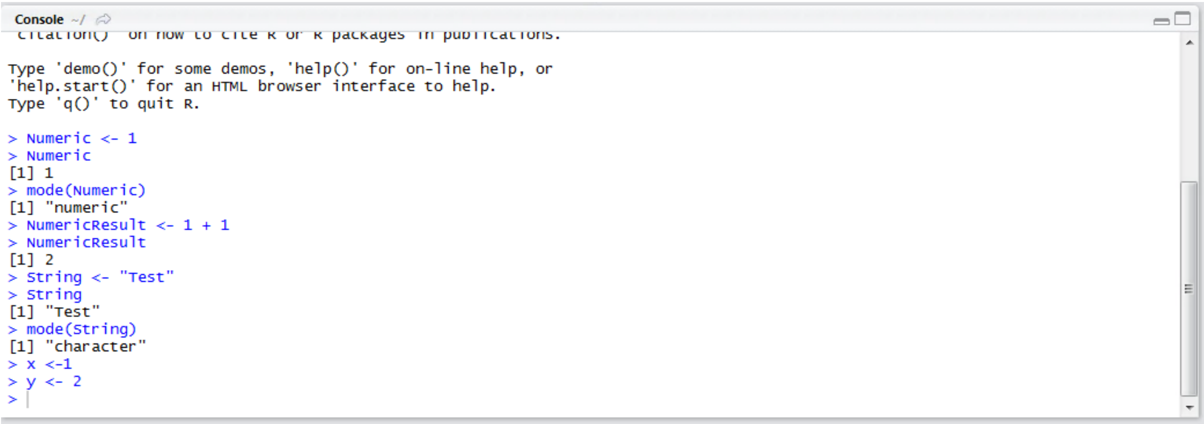


Create another variable y by typing:

y <- 2

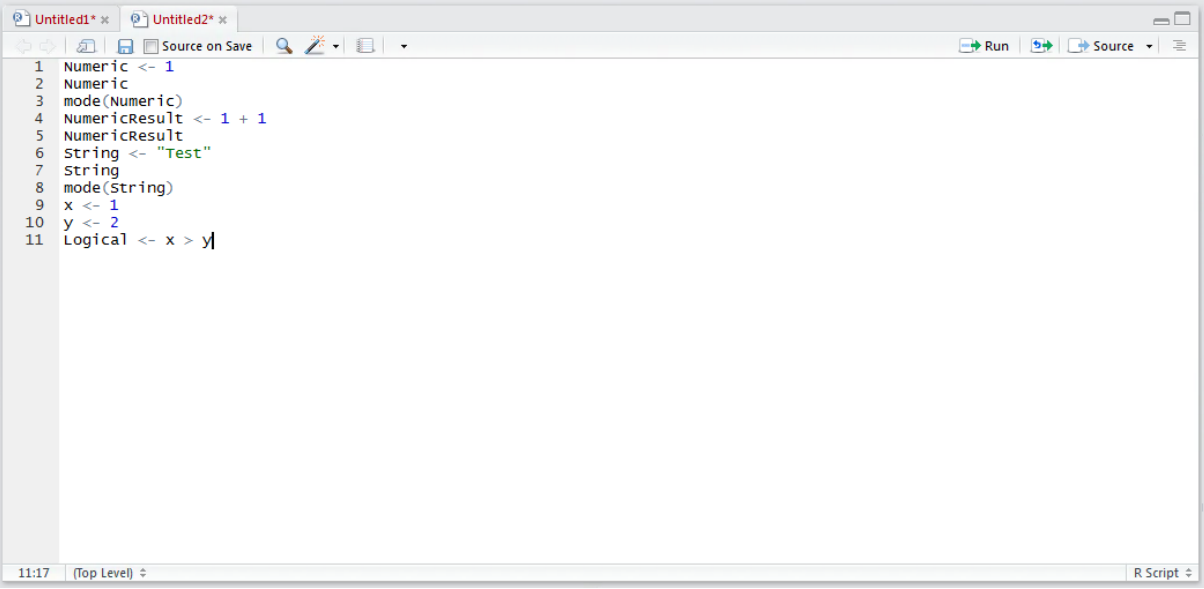


Run the line of script to console:

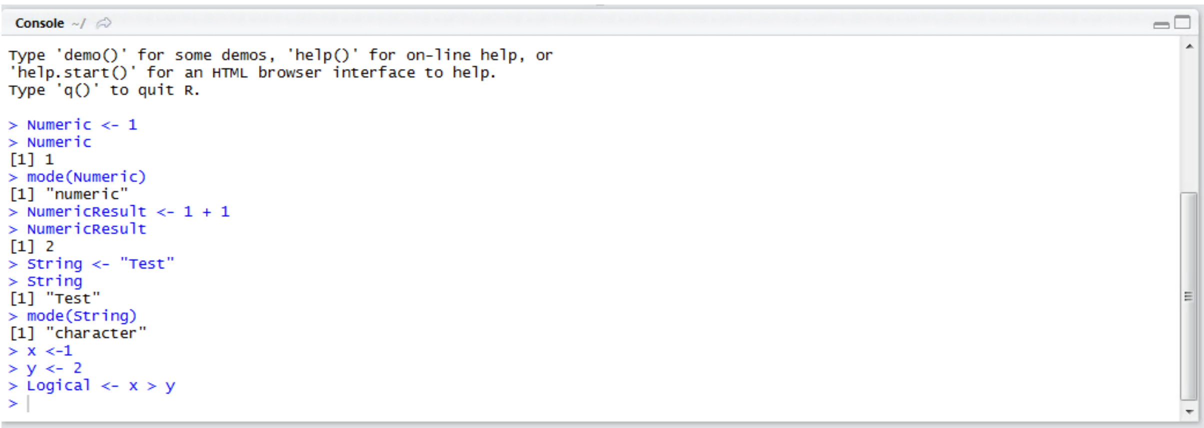


The logical variable will be created as the result of comparing one variable to another, in this case, questioning if x is greater than y. Type:

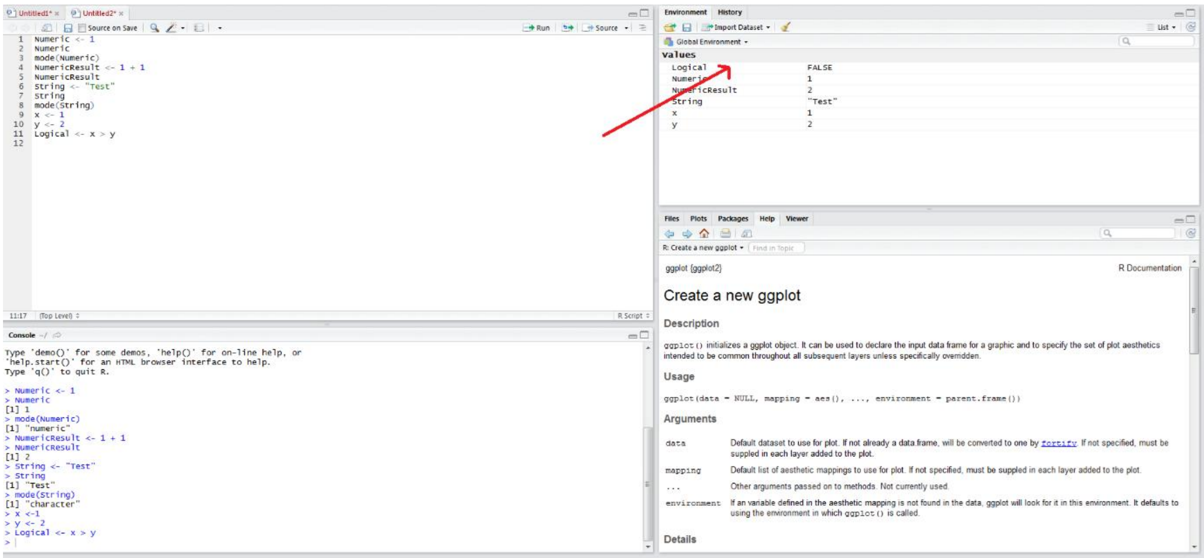
Logical <- x > y



Run the line of script to the console:

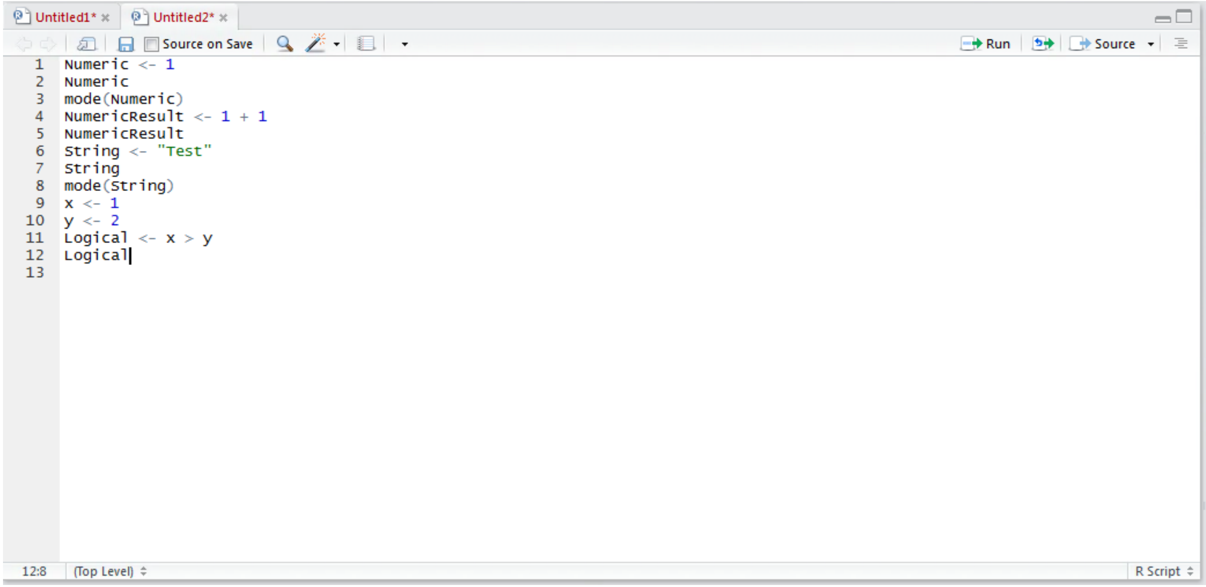


It can be seen that the variable Logical has been created and is available in the Environment pane:

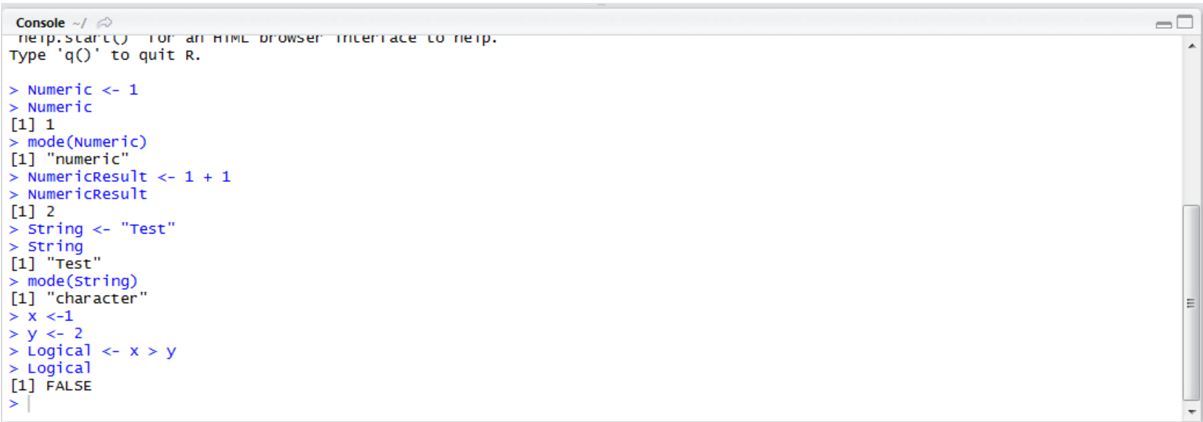


Naturally, the variable can also be referenced via simply typing into the script editor:

Logical

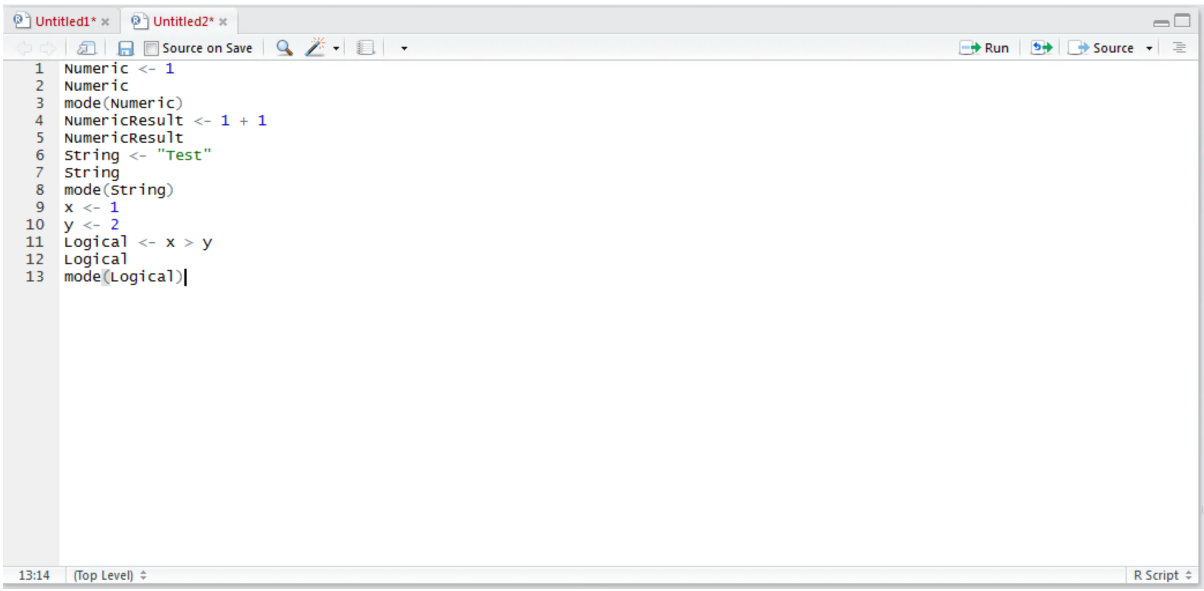


Run the script to console:



It can be seen that the variable has been written out as FALSE, in this instance, with the opposing value being TRUE. Using the mode() function, typing into the script editor:

mode(Logical)



Run the script to console:

```
> Numeric <- 1
> Numeric
[1] 1
> mode(Numeric)
[1] "numeric"
> NumericResult <- 1 + 1
> NumericResult
[1] 2
> String <- "Test"
> String
[1] "Test"
> mode(String)
[1] "character"
> x <-1
> y <- 2
> Logical <- x > y
> Logical
[1] FALSE
> mode(Logical)
[1] "logical"
>
```

It can be seen that the variable writes out as being of type logical.

Procedure 17: List Variables in R.

While RStudio will display the variables in the session at a given point in time, the function can be replicated to console also. The ls() function, which has hitherto been used to identify the functions in a package, is by default used to identify objects in the session. In the script editor, type:

ls()

```
Numeric <- 1
Numeric
mode(Numeric)
NumericResult <- 1 + 1
NumericResult
String <- "Test"
String
mode(String)
x <- 1
y <- 2
Logical <- x > y
Logical
mode(Logical)
ls()
```

Run the line of script to console:

```
Console ~/
> Numeric
[1] 1
> mode(Numeric)
[1] "numeric"
> NumericResult <- 1 + 1
> NumericResult
[1] 2
> String <- "Test"
> String
[1] "Test"
> mode(String)
[1] "character"
> x <-1
> y <- 2
> Logical <- x > y
> Logical
[1] FALSE
> mode(Logical)
[1] "logical"
> ls()
[1] "Logical"       "Numeric"       "NumericResult" "String"        "x"             "y"
>
```

The variable names are returned to the console. To reference these, it is simply a matter of typing the variable name:

String

```
Untitled1* ×   Untitled2* ×
Source on Save                                   Run    Source
Numeric <- 1
Numeric
mode(Numeric)
NumericResult <- 1 + 1
NumericResult
String <- "Test"
String
mode(String)
x <- 1
y <- 2
Logical <- x > y
Logical
mode(Logical)
ls()
String

15:7   (Top Level) ÷                                                        R Script ÷
```

Run the line of script to console:

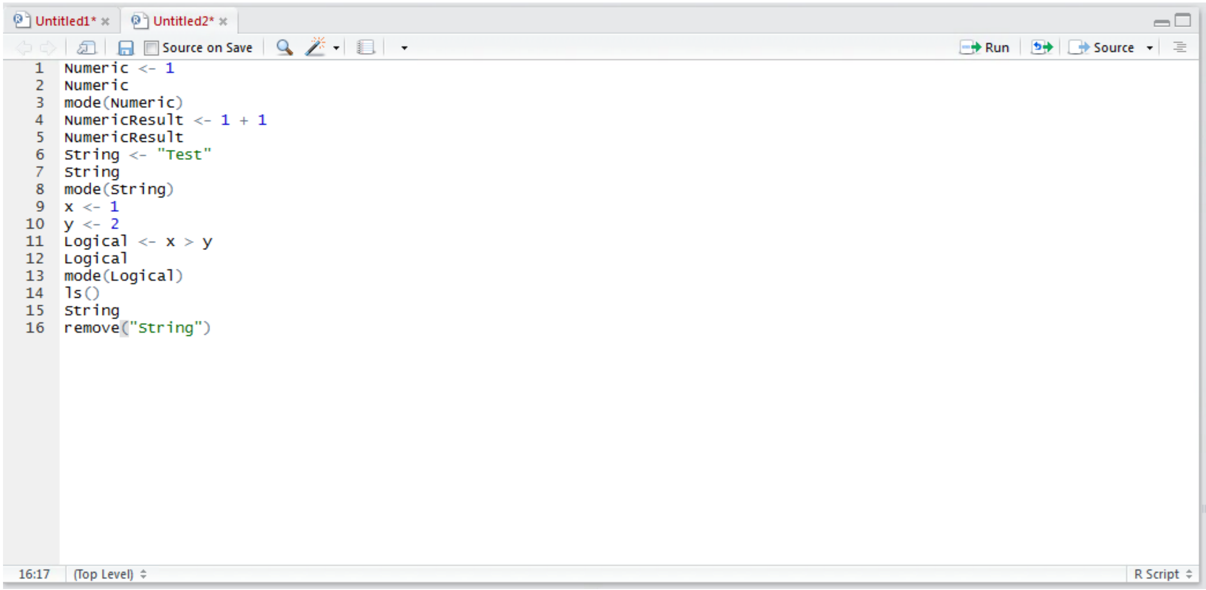
```
Console ~/
> mode(Numeric)
[1] "numeric"
> NumericResult <- 1 + 1
> NumericResult
[1] 2
> String <- "Test"
> String
[1] "Test"
> mode(String)
[1] "character"
> x <-1
> y <- 2
> Logical <- x > y
> Logical
[1] FALSE
> mode(Logical)
[1] "logical"
> ls()
[1] "Logical"       "Numeric"       "NumericResult" "String"        "x"             "y"
> String
[1] "Test"
>
```

Procedure 18: Remove Variables in R.

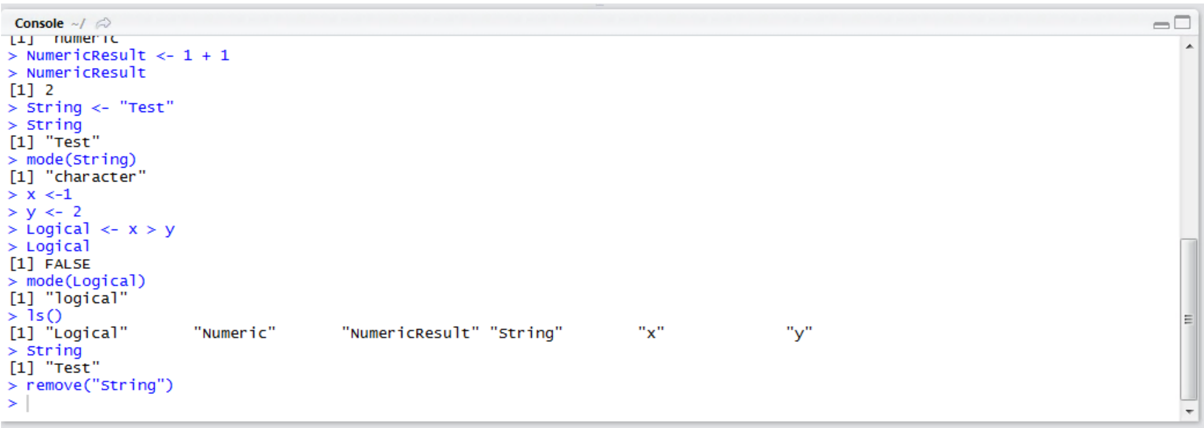
In the event that long and complex scripts are being processed, where the objects might be using a substantial amount of memory (such as a large table from a database), it may be prudent to remove the objects when the script no longer needs it.

To remove an object, the remove() function is used taking an argument as the name of the variable to be removed. In this example, the String variable will be removed. Type:

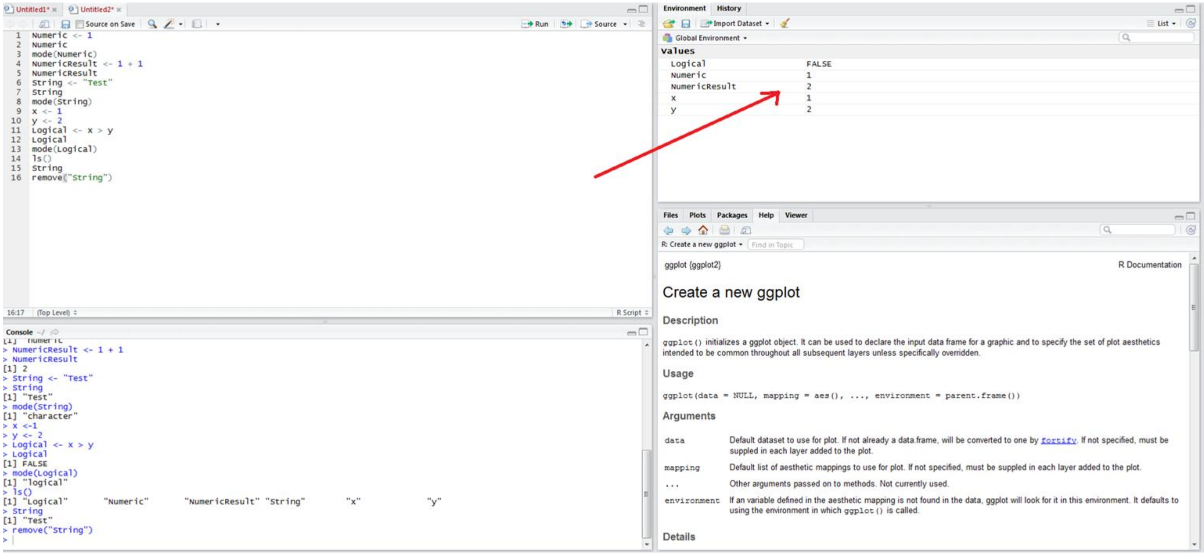
remove("String")



Run the line of script to console:

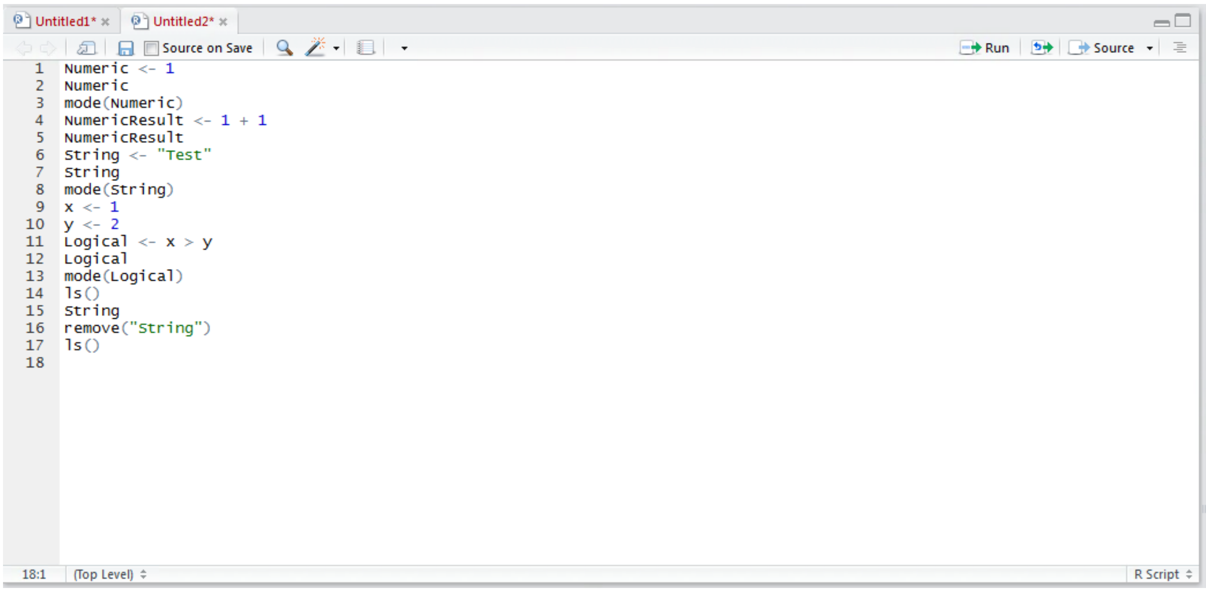


It can be seen that the String variable no longer appears in the environment pane:



Naturally the variable will not be available in the session upon inspection of the ls() function. Type:

ls()



Run the line of script to console:

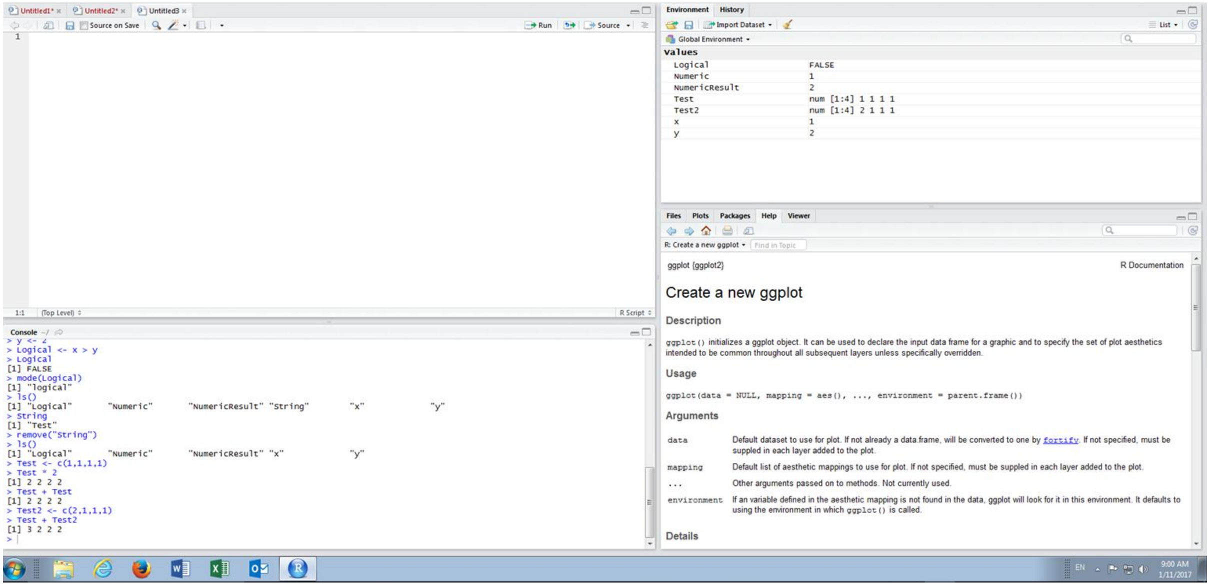
It can be observed that the return is now minus the String variable.

Module 3: Data Structures Introduction.

Although R seems intimidating at first, requiring what seems to be programming skills, this belies that most of the procedures for complex predictive analytics can in fact be distilled into simple procedures. It is most certainly not correct that R need be viewed upon as a programming language.

There are certain basic principles that need to be understood however and as covered in Module 1, Module 2 sets out to emphasise these principles.

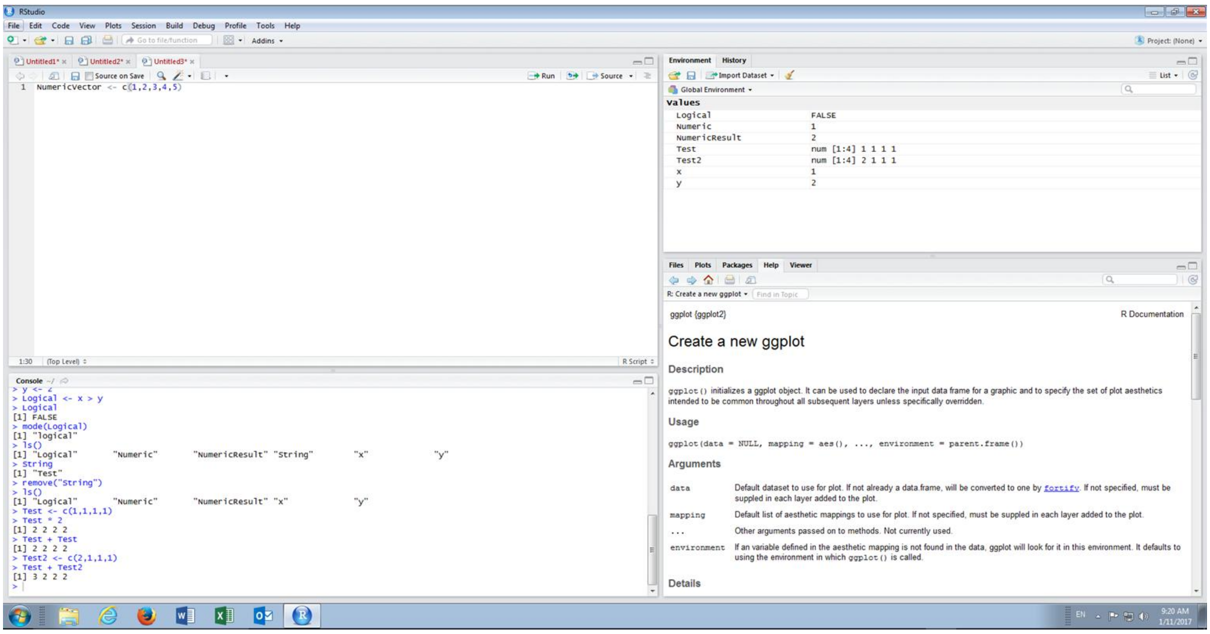
In this module, Data Structures, available to R, will be explored. The exercise will require a new script to have been opened in RStudio as will have become familiar in procedures set forth in Module 1:



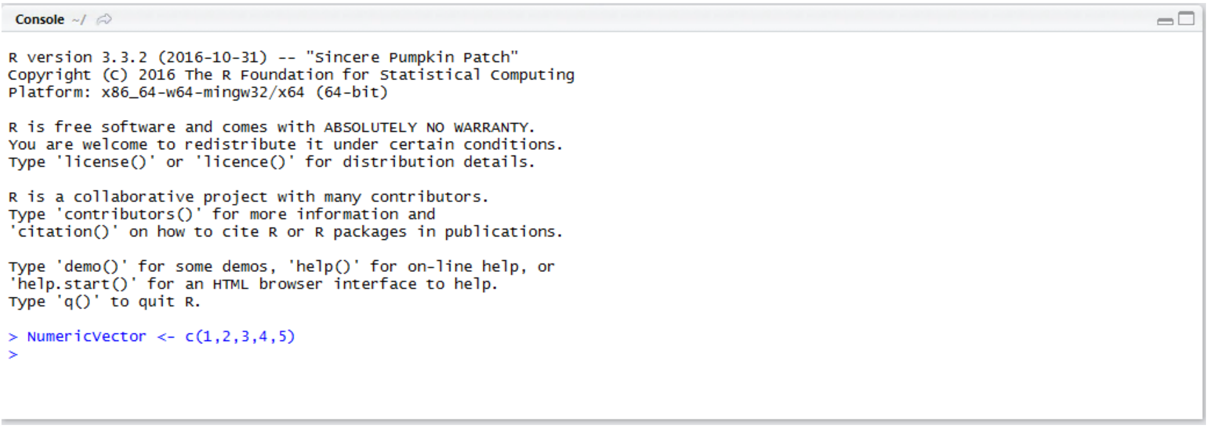
Procedure 1: Create a Vector with c Function.

The c function is used to combine variables into a vector. To create a numeric Vector, start by typing:

NumericVector <- c(1,2,3,4,5)

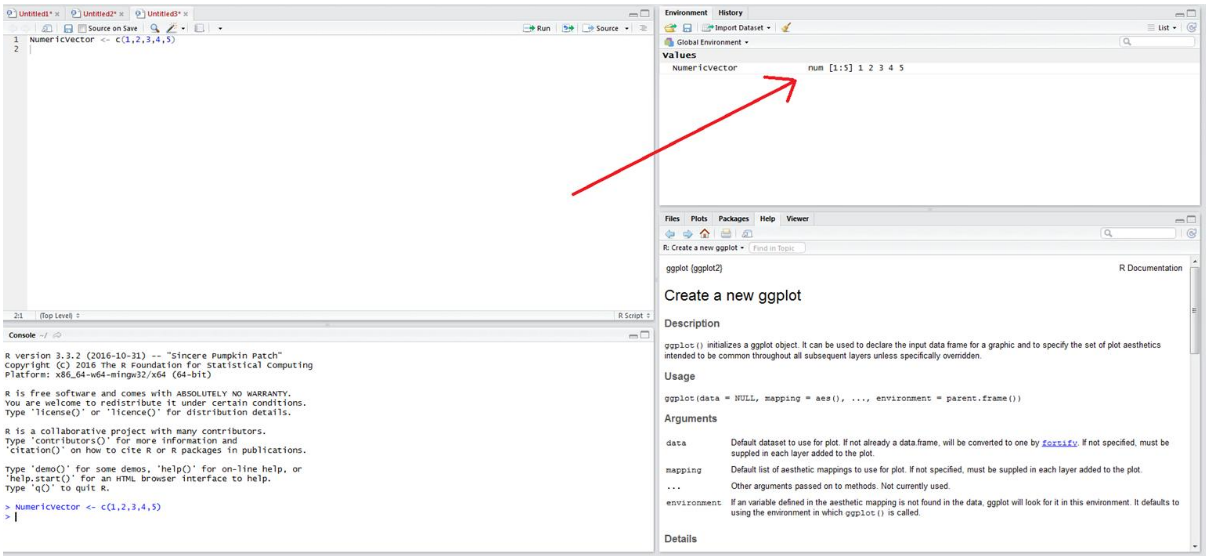


Run the line of script to console:



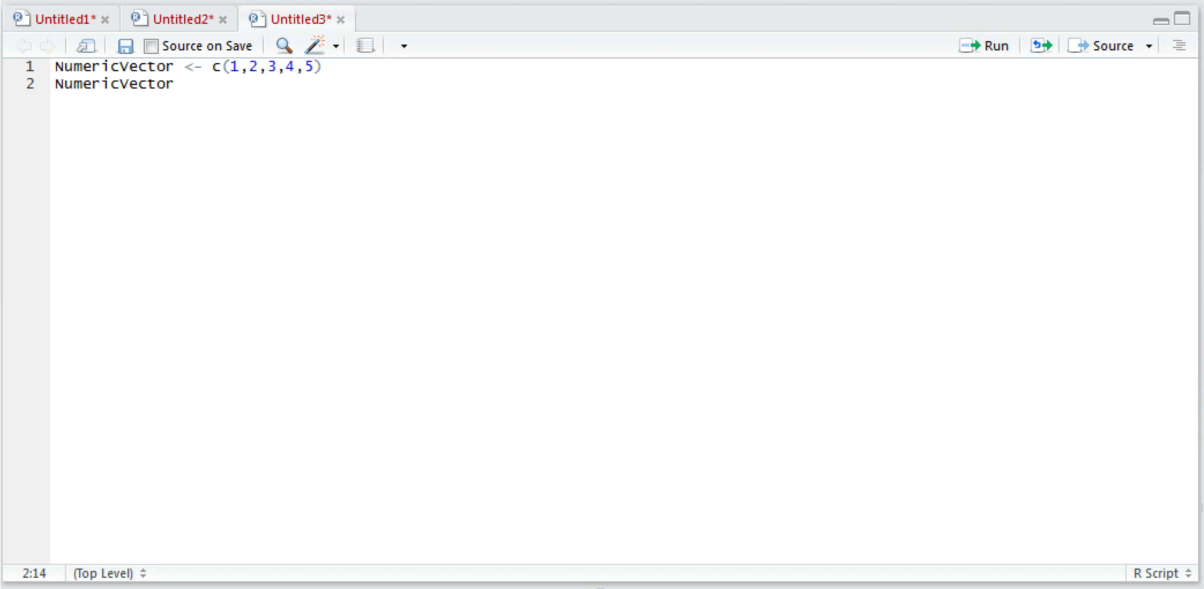
```
Console ~/

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> NumericVector <- c(1,2,3,4,5)
>
```

The vector appears in the envirponment pane, showing the dimensons of [1,5], which would suggest 1 row, five columns:

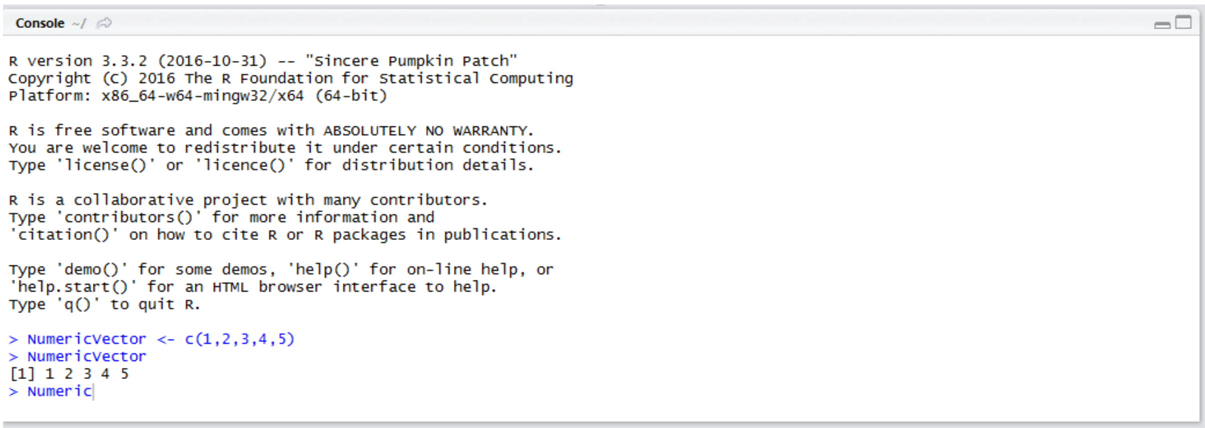


The vector can be referenced in the console, as with all other variables, by typing:

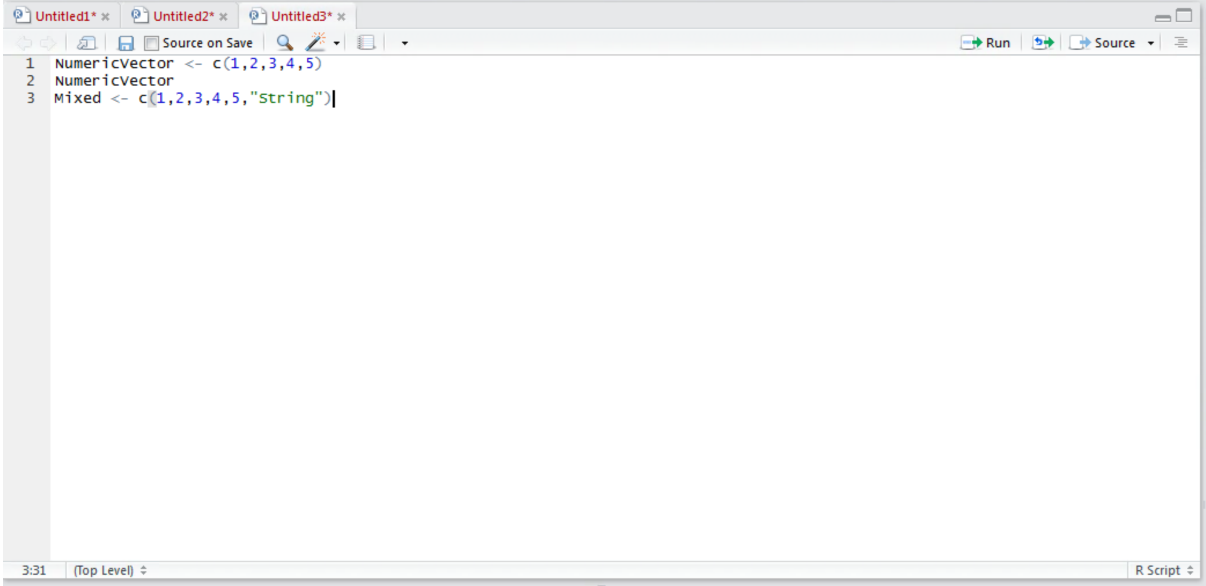
NumericVector



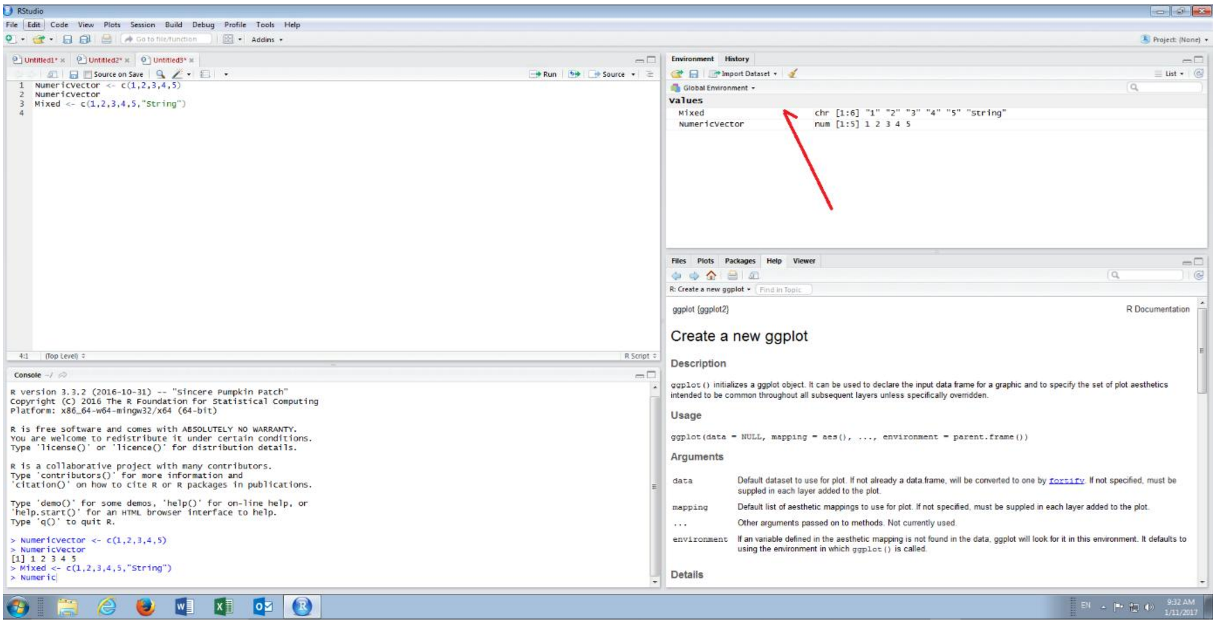
Run the line of script to the console:



To observe how R handles vectors, comprised of separate types (in so far as it CANT handle it), start by typing:

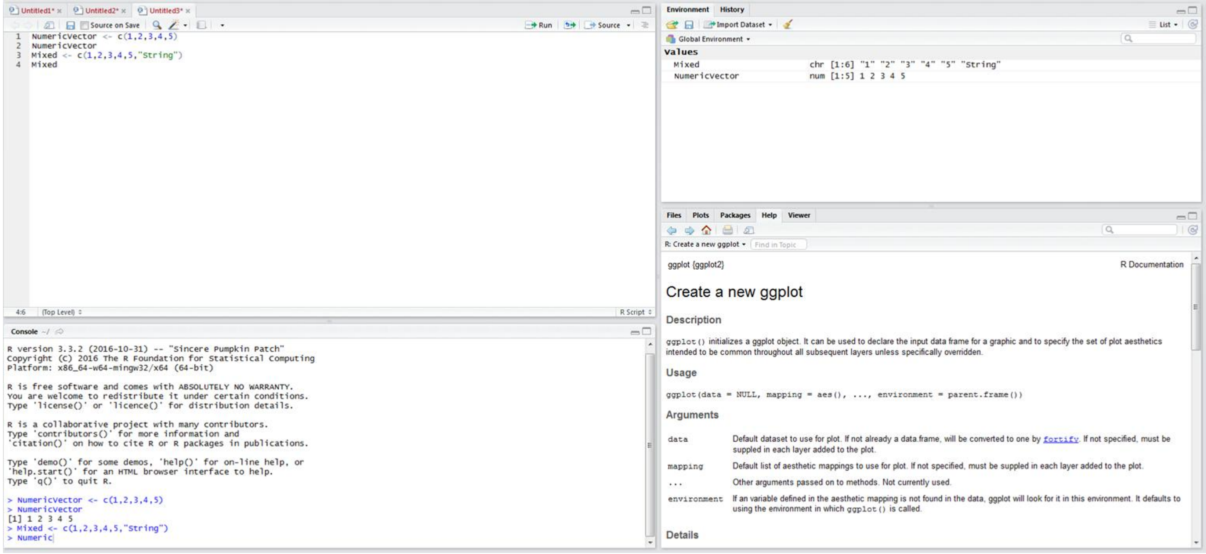


Run the script to console:

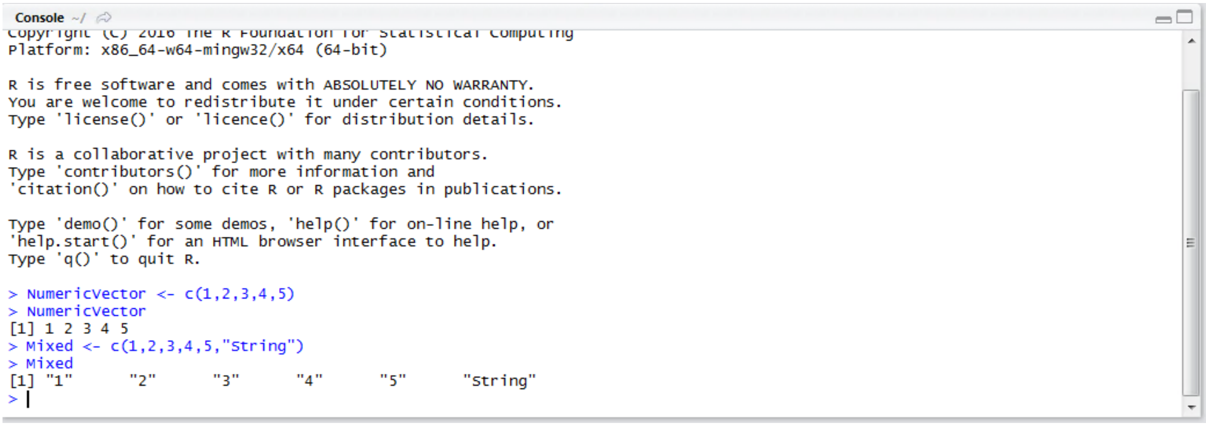


It can be seen that the vector has been created and is displayed in the environment pane, however, it is being created as a character vector owing to the presence of character argument which cannot be coerced to a numeric value and as such the entire vector becomes a character vector. To validate this in the console, type:

Mixed



Run the line of script to console:



It can be validated that the vector has been created as a string, based on the premise of the double quotations around all of the entries.

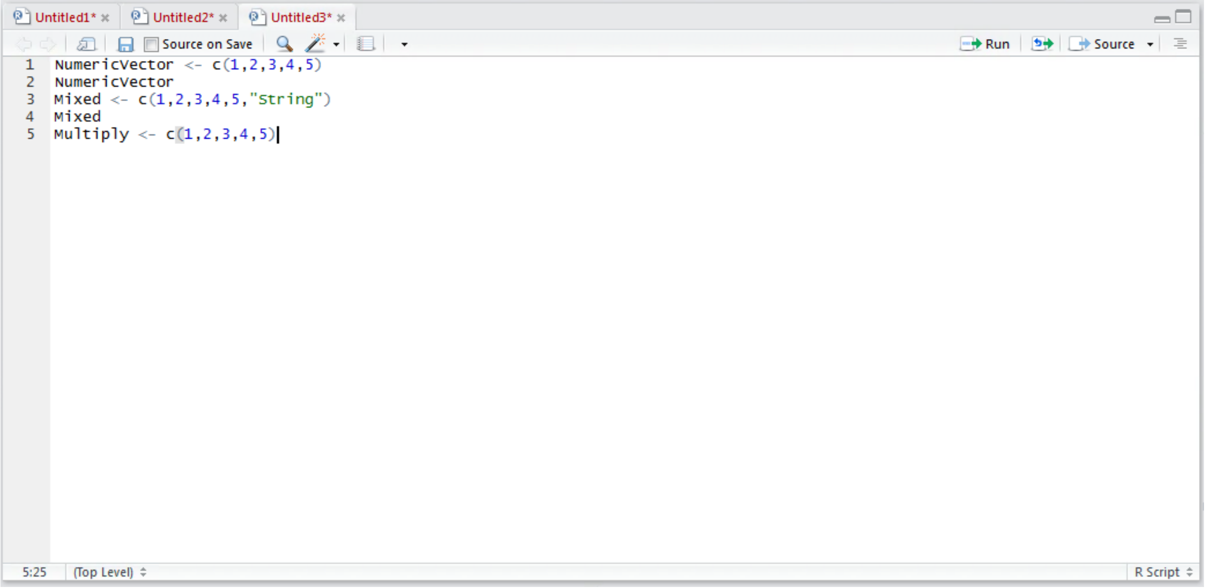
Procedure 2: Perform Vector Arithmetic.

A variety of arithmetic operators can be used against vectors such as:

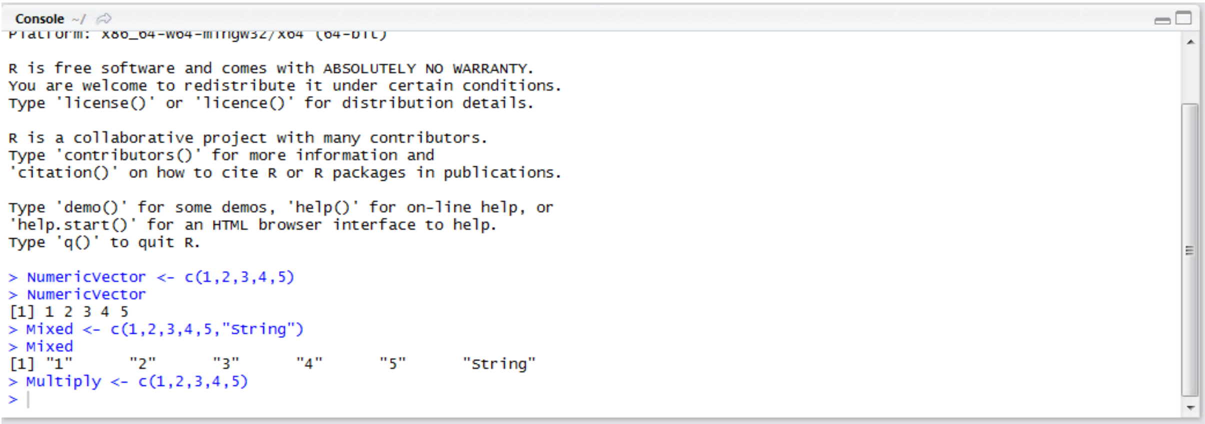
- + Addition
- - Subtraction
- * Multiplication
- / Division
- ^ Power
- %% mod

In this example, a numeric Vector will be multiplied by 2. Start by creating a Vector, type:

Multiply <- c(1,2,3,4,5)

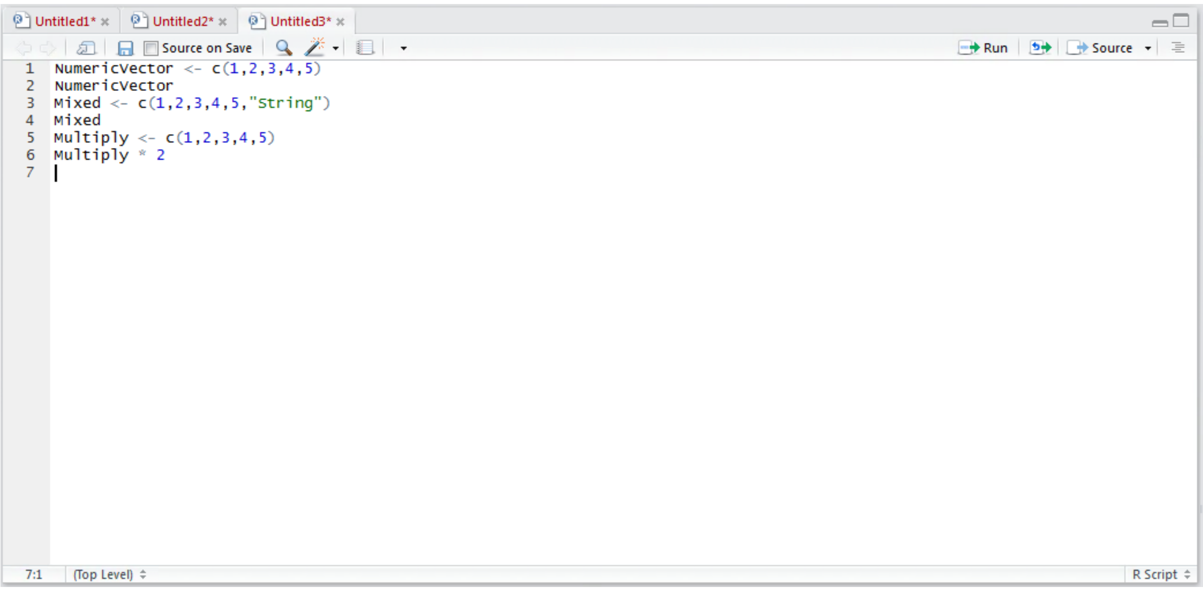


Run the line of script to console:

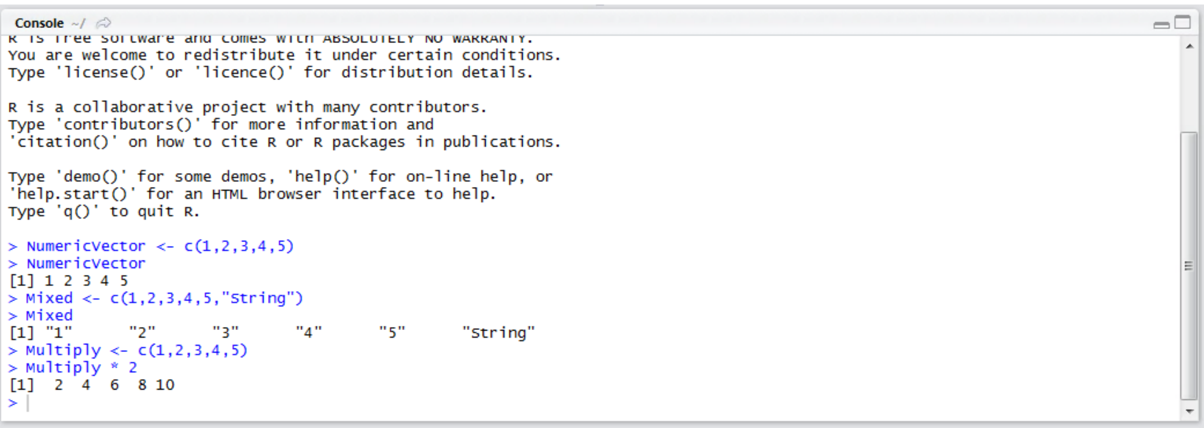


In this example, multiply the vector by 2. Type:

Multiply * 2

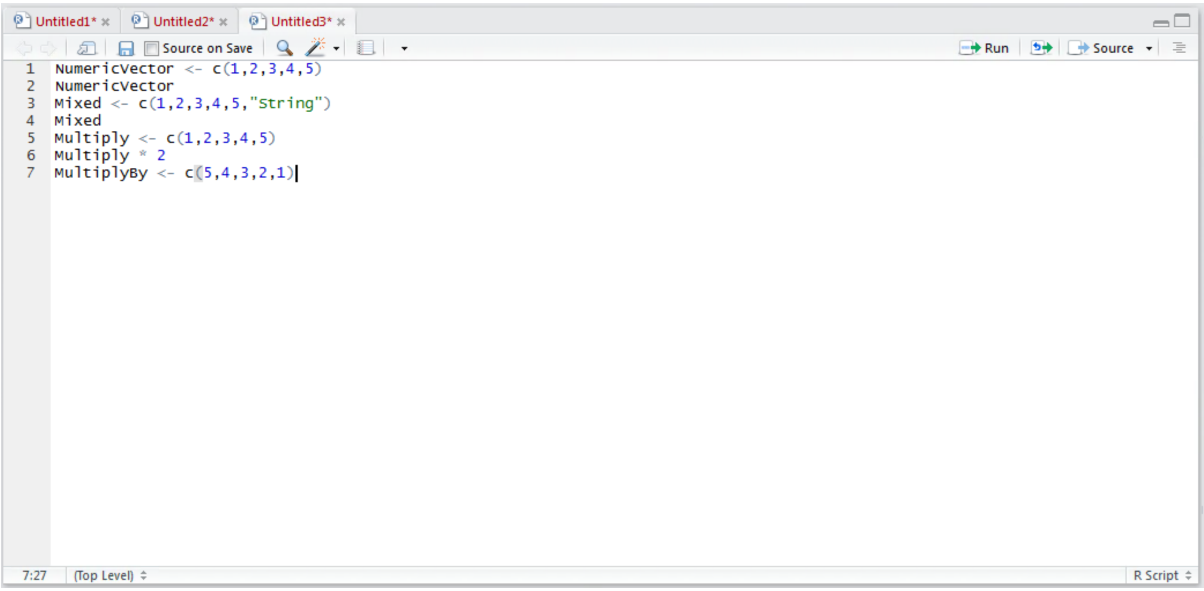


Run the line of script to console to write out the new vector:



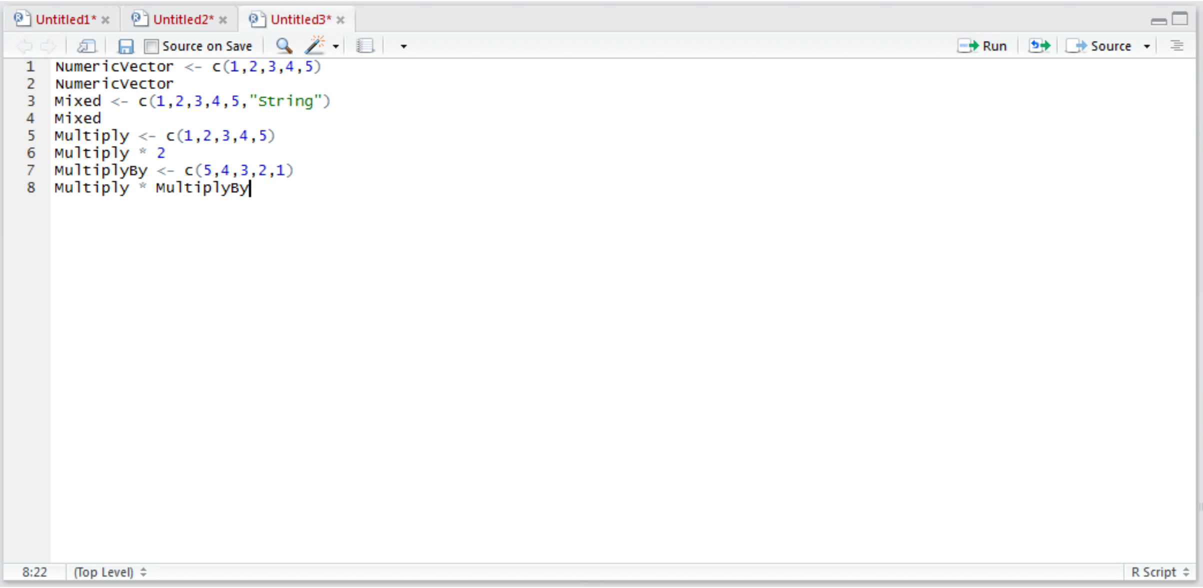
It can be observed that each position in the vector has been multiplied by the value of 2. It is also possible to multiple by another vector. Create another vector by typing:

MultiplyBy <- c(5,4,3,2,1)

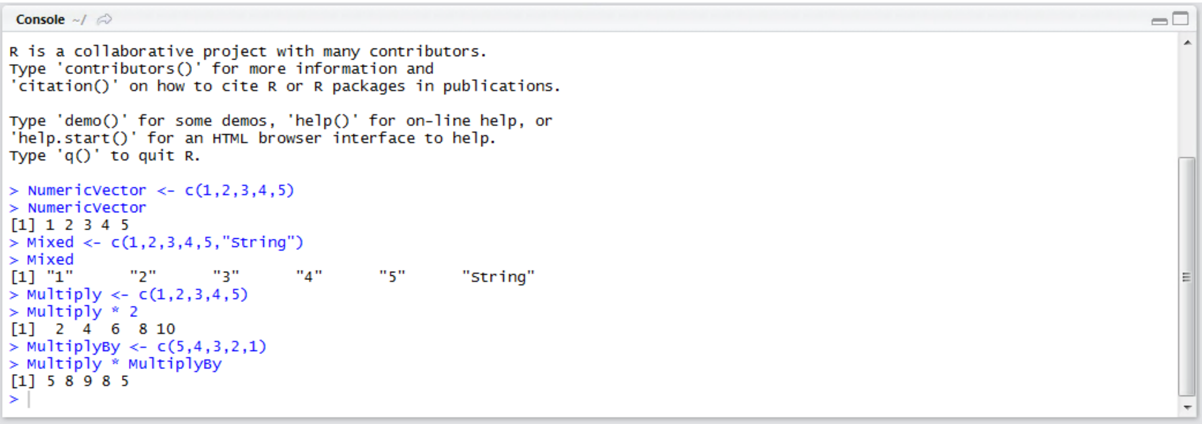


Then multiply the existing vector Multiply by the new vector MultiplyBy by typing:

Multiply * MultiplyBy



Run the line of script to console:



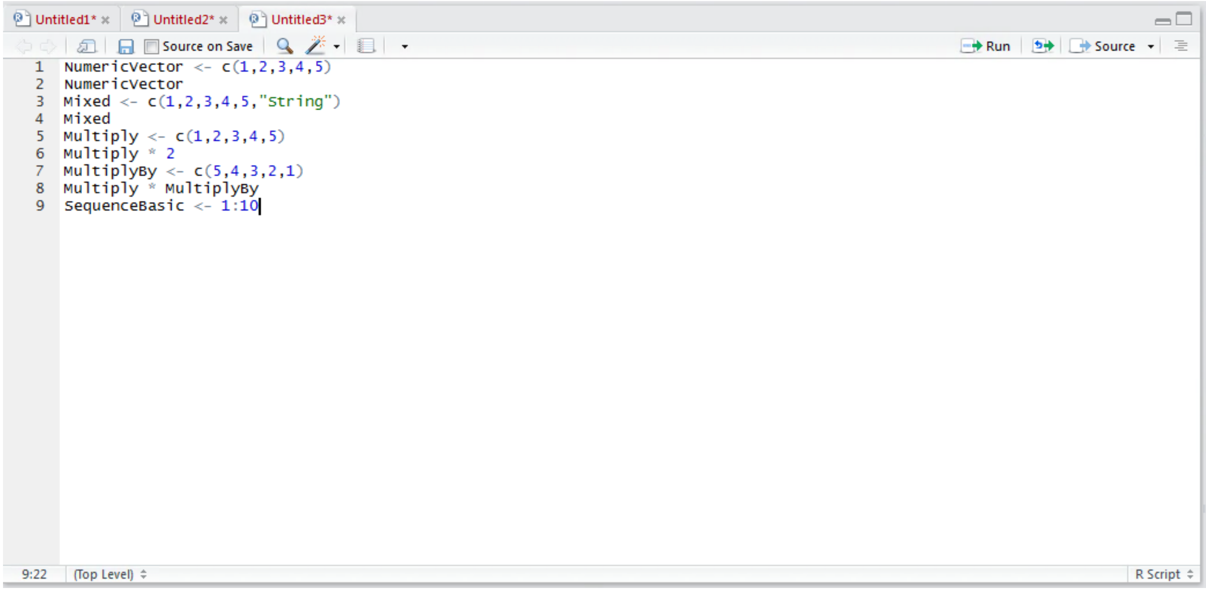
It can be observed that for each position in the vector, the value in that position has been multiplied by the same position in the other vector. Think of this as the equivalent of filling down in an Excel spreadsheet.

Procedure 3: Create Vector via a Sequence.

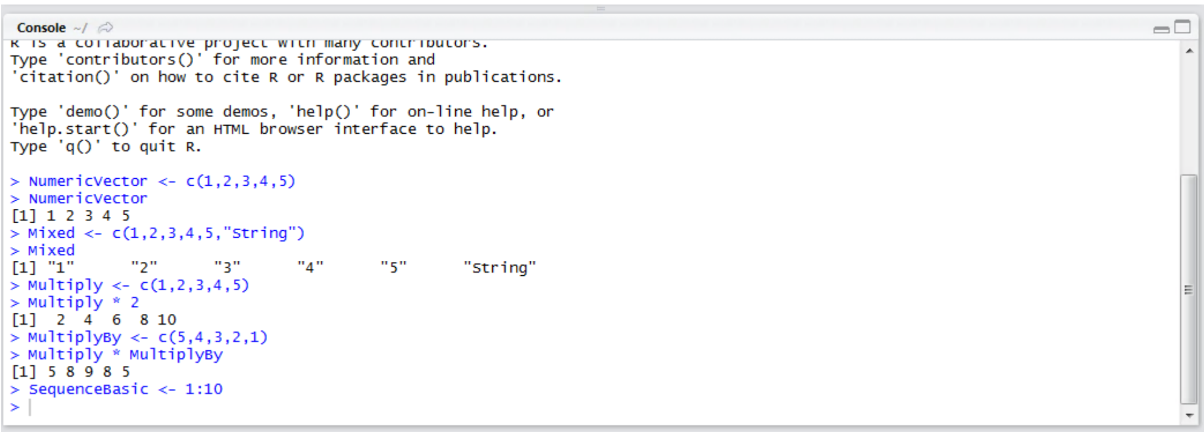
There are two main ways to create vectors in a sequence of number in R, the first is using the semicolon in assignment, the second is using a function that archives much the same while offering more flexibility. The purpose of this procedure is to introduce some of the more sophisticated elements of the R language, however, for the purposes of predictive analytics it is not absolutely necessary to delve into such depth to achieve the end result of reliable predictive analytics.

To create a vector which is a sequence of numbers from 1 to 10, type:

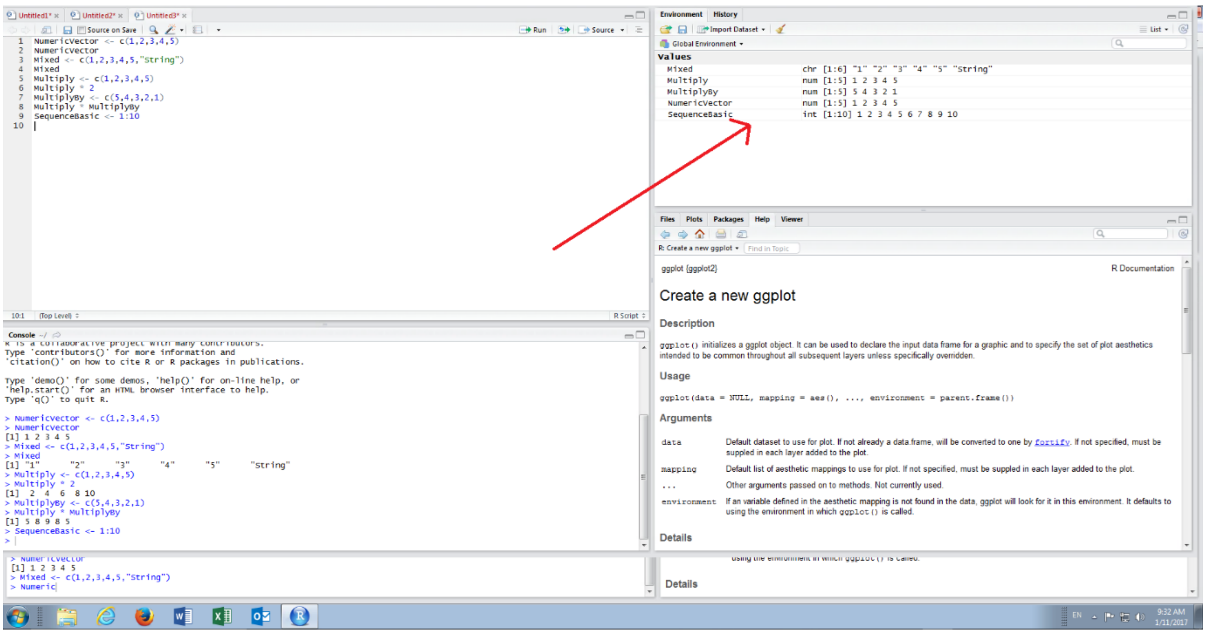
SequenceBasic <- 1:10



Run the line of script to console:

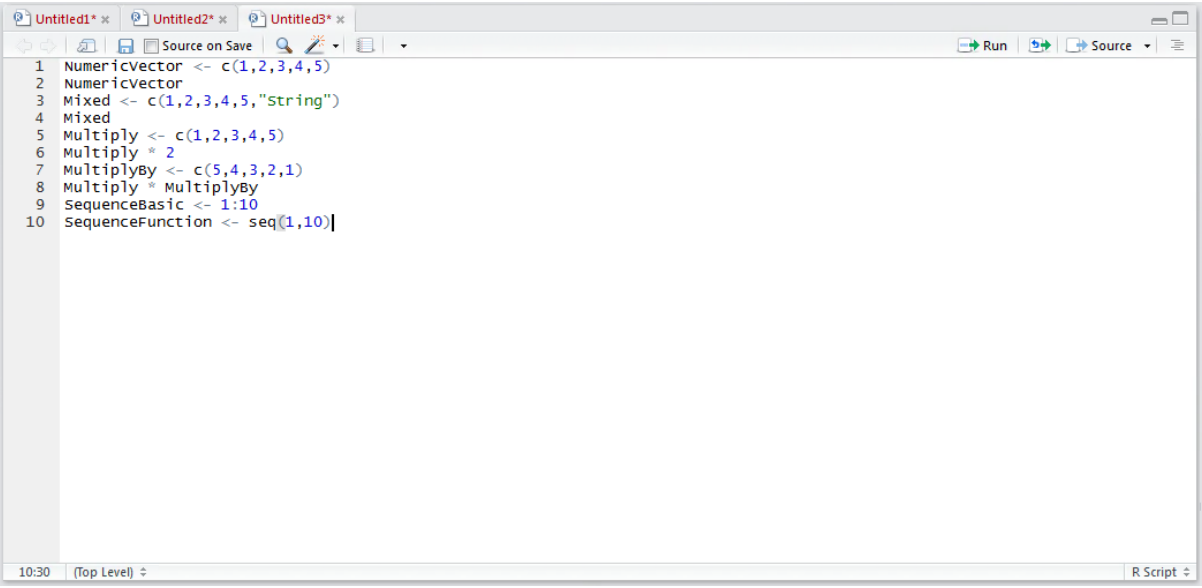


It can be seen from the environment pane that the vector has been created and that the values span from 1 to 10 in increments of 1:

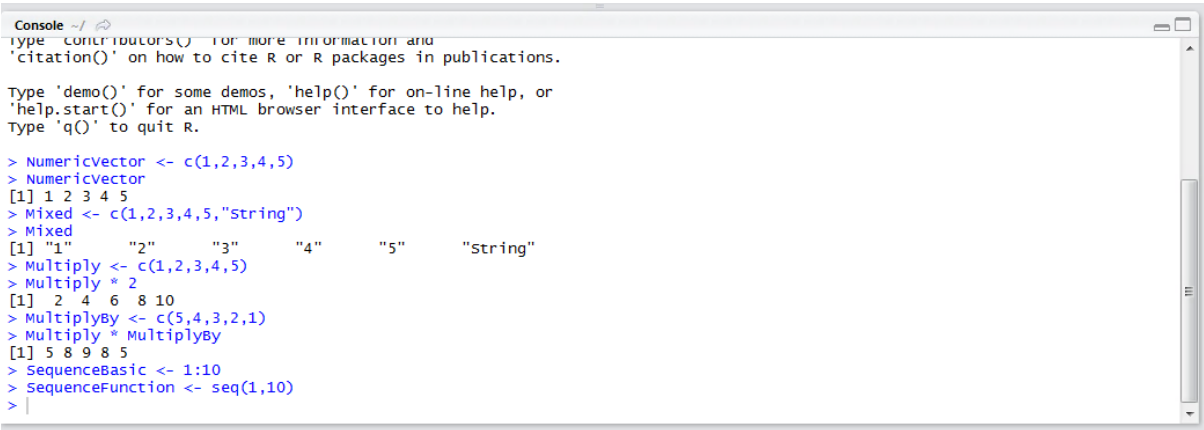


Introducing functions, the same using the seq() function can be achieved by typing:

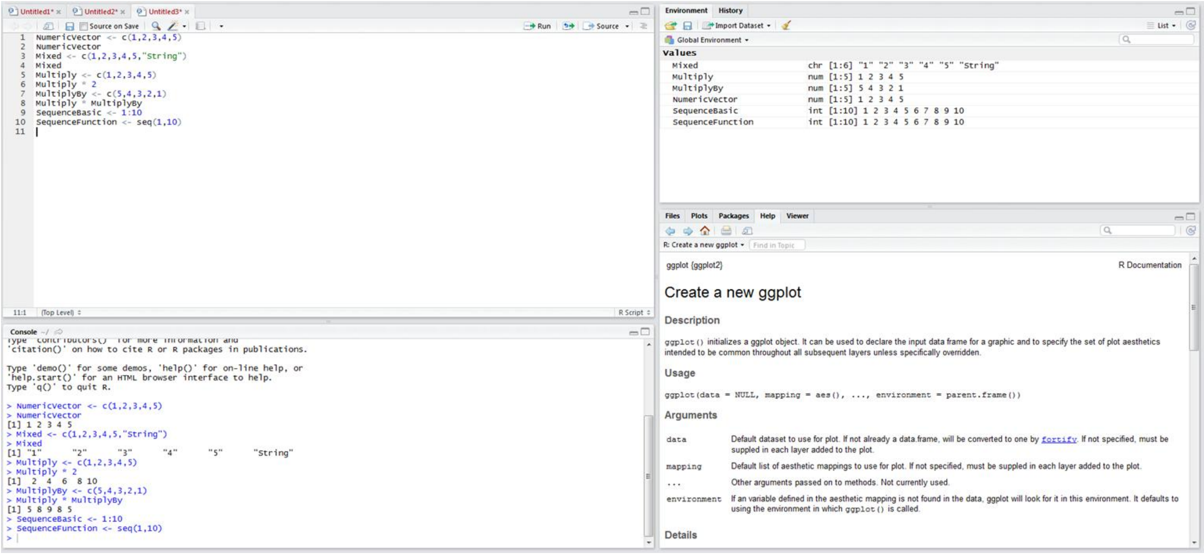
SequenceFunction <- seq(1,10)



Run the line of script to console:

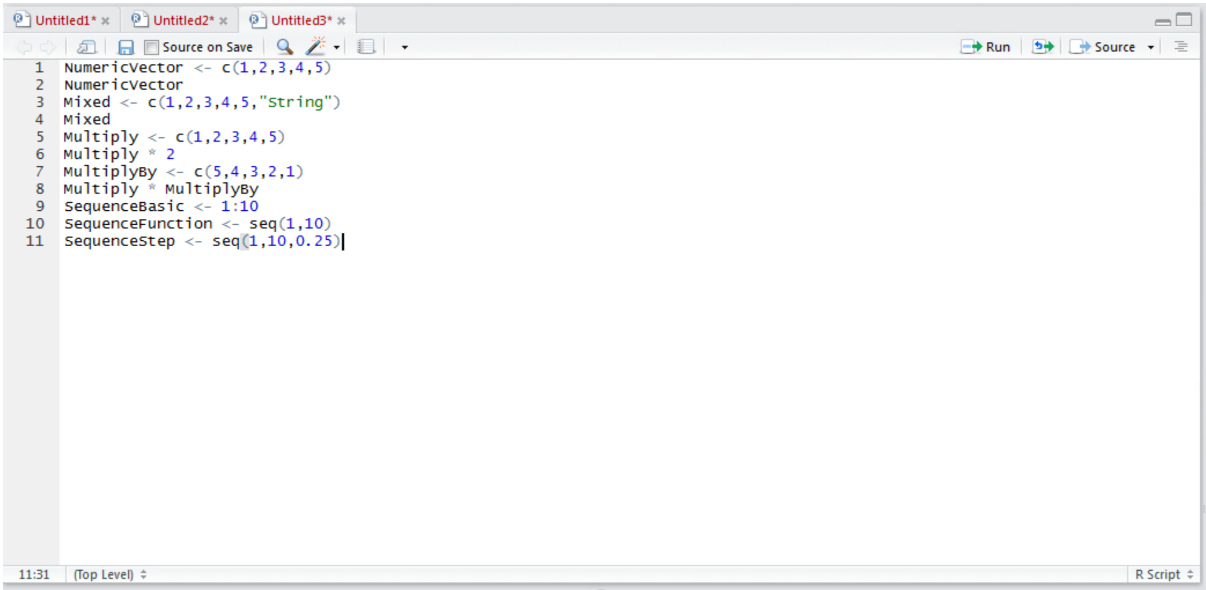


It can be observed that SequenceBasic and SequenceFunction take the same form in the environment pane:

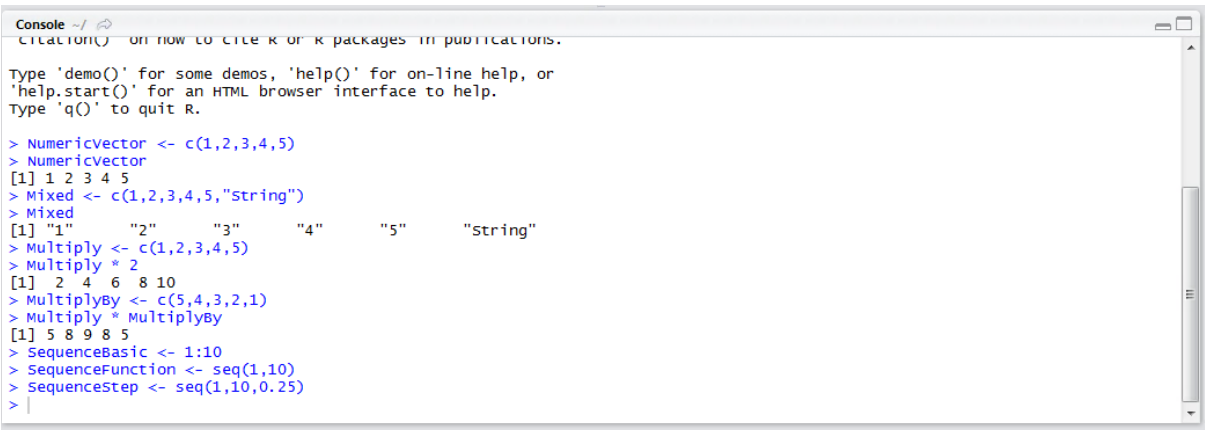


The benefits of using the seq() function is that it allows for sequences to be created with different steps sizes, where the default is 1. To create a step of 0.25, type:

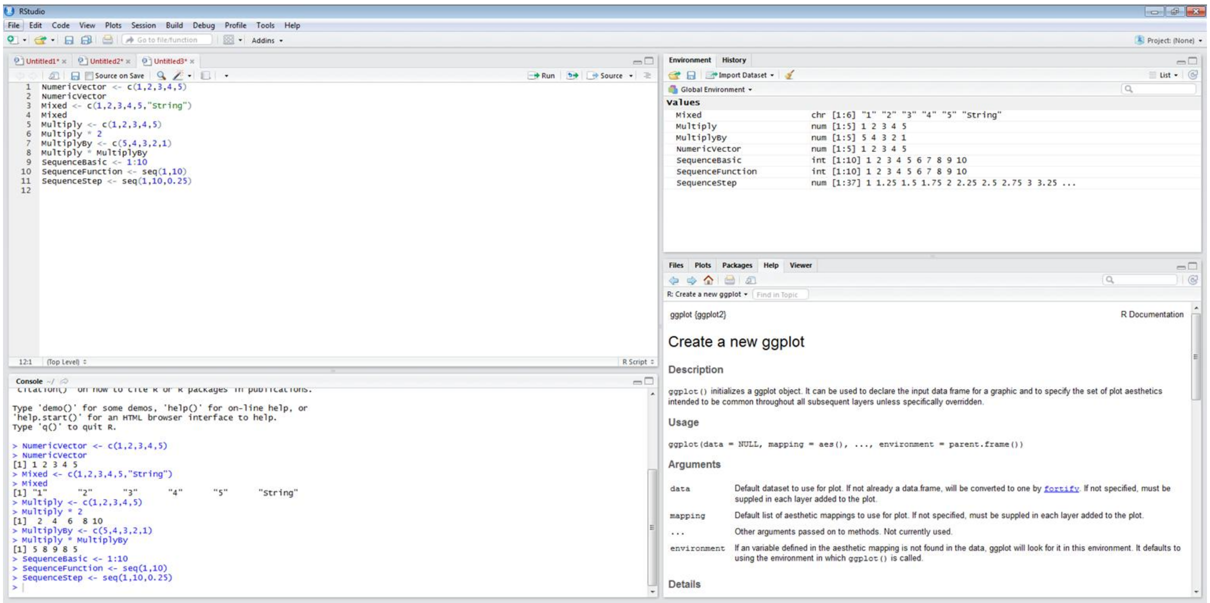
SequenceStep <= seq(1,10,0.25)



Run the line of script to console:



It can be seen that a much larger vector has been created by inspecting the environment pane, where the values increase by 0.25 increments:

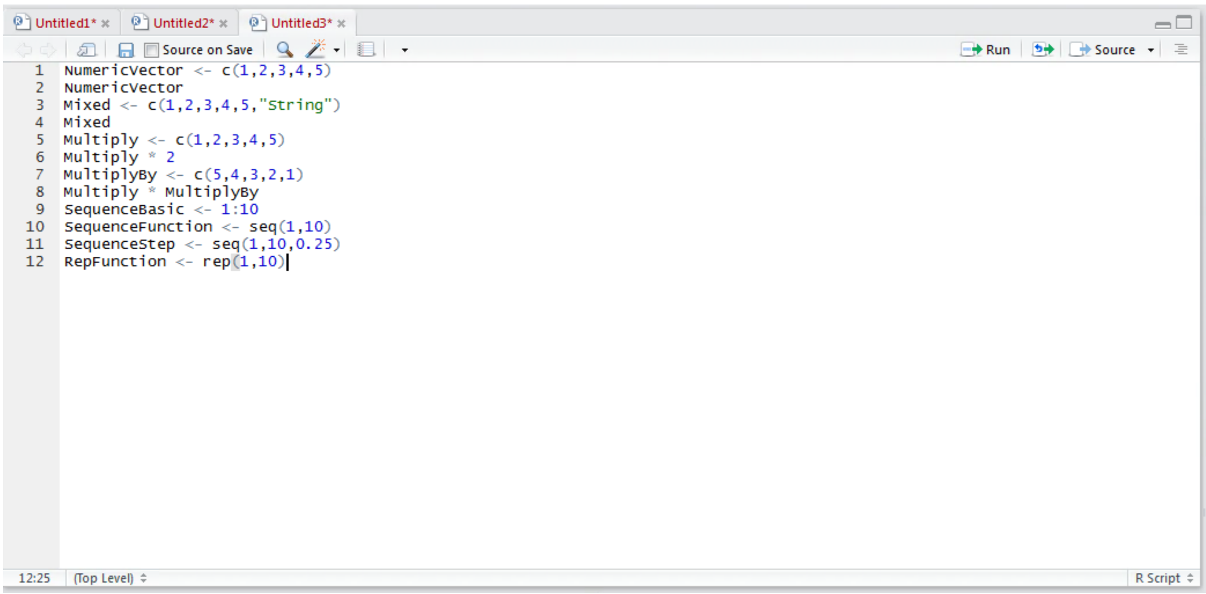


The seq() function provides a lot of other options for the creation of sequences such as repetition which would be outside the scope of this procedure. The seq() function has been introduced as a means to demonstrate assignment by function return values.

Procedure 4: Create a Vector via Repetition.

Hitherto repetition of values in a vector has been achieved by typing out the vector using the c() function (i.e c(1,1,1,1,1,1,)). The rep() function can achieve this quite simply, by taking the value and then an argument specifying the number of times this is to be repeated:

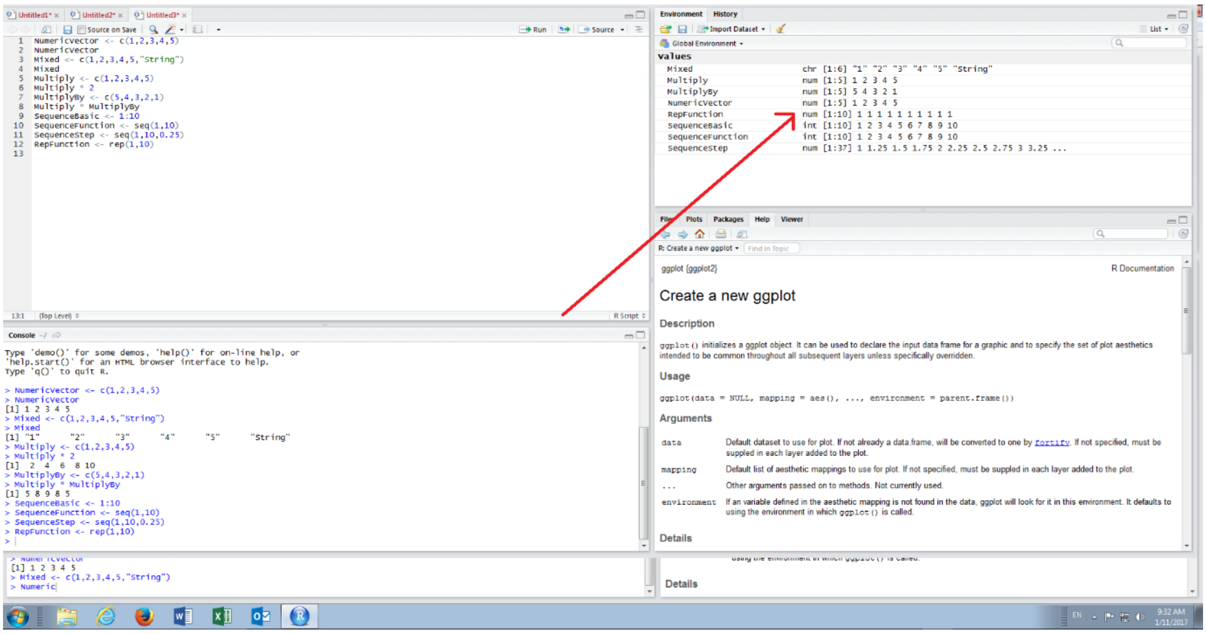
RepFunction <- rep(1,10)



Run the line of script to console:

```
Console ~/
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> NumericVector <- c(1,2,3,4,5)
> NumericVector
[1] 1 2 3 4 5
> Mixed <- c(1,2,3,4,5,"String")
> Mixed
[1] "1"       "2"       "3"       "4"       "5"       "String"
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> |
```

It can be observed in the environment pane that a vector has been created, repeating the value 1, 10 times:



As with the seq() function, the rep() function provides many more options which are outside the scope of this procedure.

The rep() function is used most commonly in these procedures for the purposes of creating dummy variables in Data Frames, where it may be called upon to add a vector to a Data Frame yet it is imperative to create the vector manually via the c function owing to the possibility that there is many thousands of entries.

Procedure 5: Selecting and Filtering from a numeric Vector.

There are a number of ways to specifically extract data from a vector, a process sometimes called subscripting. In this procedure, the vector created in procedure 21 will be used. The simplest way to extract data from a vector is to specify the position inside square brackets. To subscript and retrieve the third value in the vector type:

SequenceBasic[3]

```r
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
```

Run the line of script to console:

```
help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> NumericVector <- c(1,2,3,4,5)
> NumericVector
[1] 1 2 3 4 5
> Mixed <- c(1,2,3,4,5,"String")
> Mixed
[1] "1"      "2"      "3"      "4"      "5"      "String"
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
>
```

It can be observed that the value at the third position in the SequenceBasic vector has been returned. Alternatively, specifying a negative value of 3 would return everything except the third position:

```r
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
```

Run the line of script to console:

```
Console ~/
> NumericVector <- c(1,2,3,4,5)
> NumericVector
[1] 1 2 3 4 5
> Mixed <- c(1,2,3,4,5,"String")
> Mixed
[1] "1"      "2"      "3"      "4"      "5"      "String"
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
>
```

It can be observed that the third position of the vector has been excluded in the output.

Far more powerful is the ability to select from vectors based upon a logical statement, such as all values > 5:

SequenceBasic[SequenceBasic > 5]

```
Untitled1* × Untitled2* × Untitled3* ×
Source on Save                                   Run   Source
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]

16:1   (Top Level) ÷                                   R Script ÷
```

Run the line of script to console:

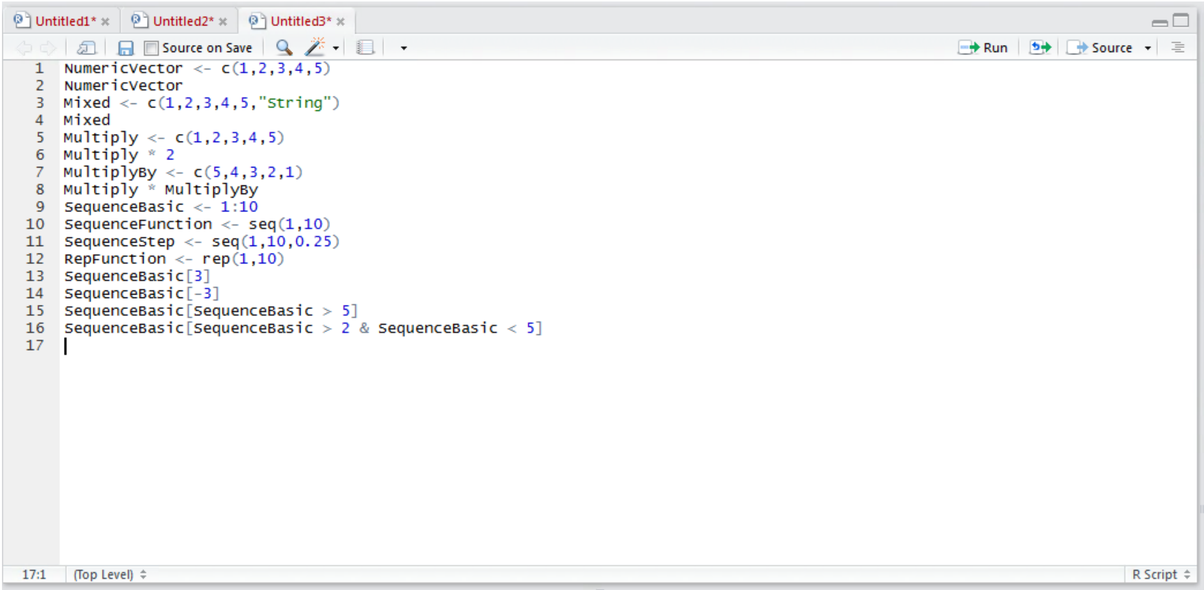
```
Console ~/
> NumericVector
[1] 1 2 3 4 5
> Mixed <- c(1,2,3,4,5,"String")
> Mixed
[1] "1"      "2"      "3"      "4"      "5"      "String"
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
> SequenceBasic[SequenceBasic > 5]
[1]  6  7  8  9 10
>
```

It can be seen that only values greater than five have been returned. The notion of selecting from a vector based on logical conditions further introduces operators:

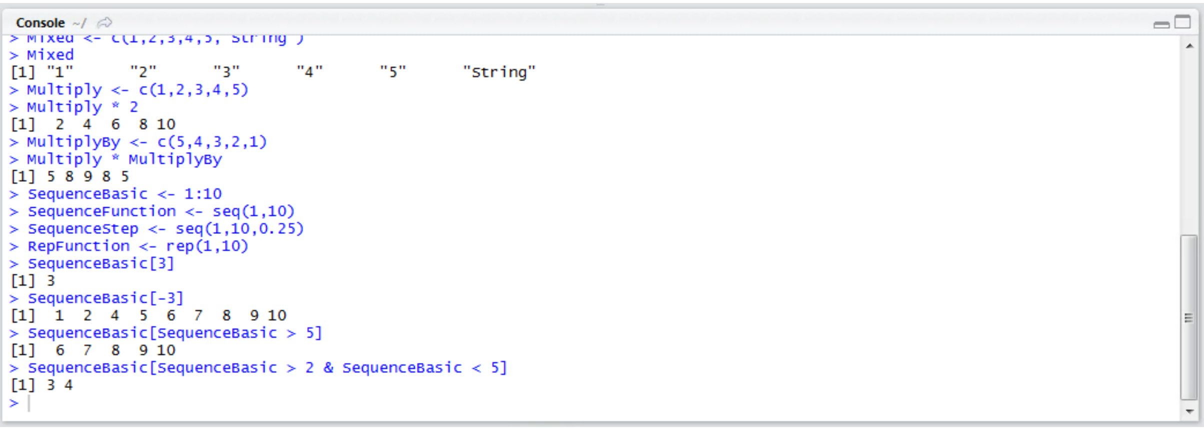
- & And.
- | Or.
- ! Not.

To create more discriminating selection from a vector, where the value must be > 2 and less than 5, type:

SequenceBasic[SequenceBasic > 2 & SequenceBasic <5]



Run the line of script to the console:



It can be seen that only the two values between 2 and 5 have been returned.

Procedure 6: Setting Vector Labels \ Names.

Selecting from a chracter vector follows the same pattern, in so far as the crieria sits inside [] square brackets and allows for the specifc selection of values or the specific exclusion of values. Create a chracter vector by typing numbers, hencforth ages:

Ages <- c(22,23,28)

```
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
```

Run the line of script to console:

```
> Mixed
[1] "1"      "2"      "3"      "4"      "5"      "String"
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
> SequenceBasic[SequenceBasic > 5]
[1]  6  7  8  9 10
> SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
[1] 3 4
> Ages <- c("22","23","28")
>
```

It is possible to add labels to the entries in the vector using the names() function, similar to column headers in an Excel spreadsheet. The label 22 is Tom's Age, 23 is Harries Age and lastly 28 is Dicks Age. To add labels to each Vector value, type:

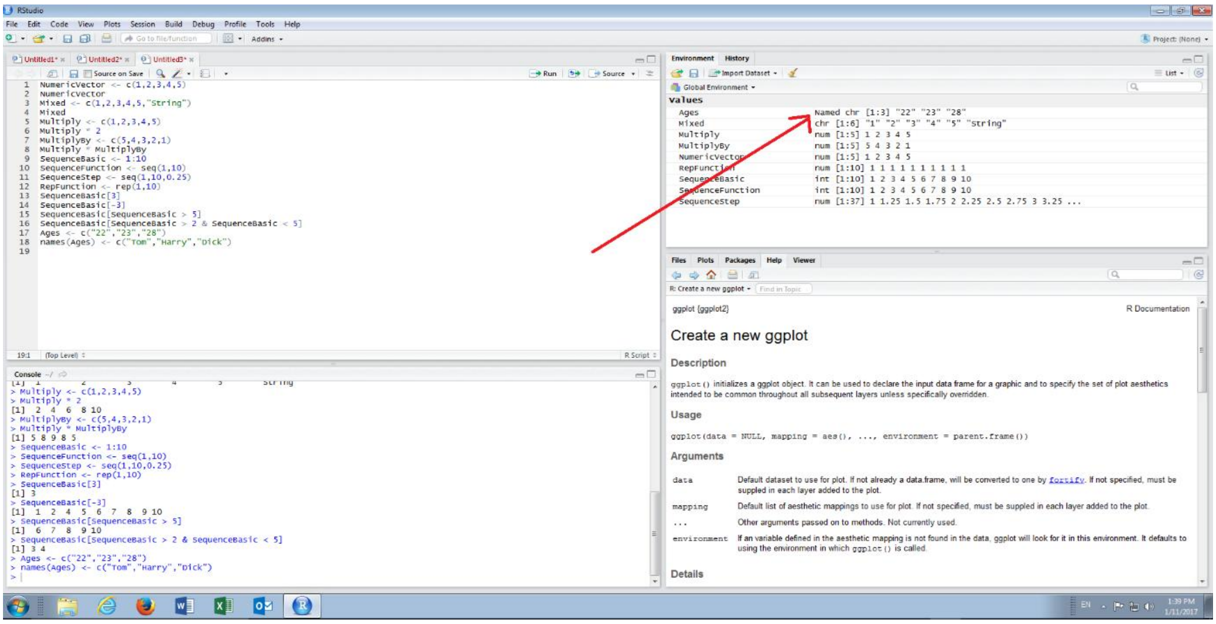
names(Ages) <- c("Tom","Harry","Dick")

```
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
```

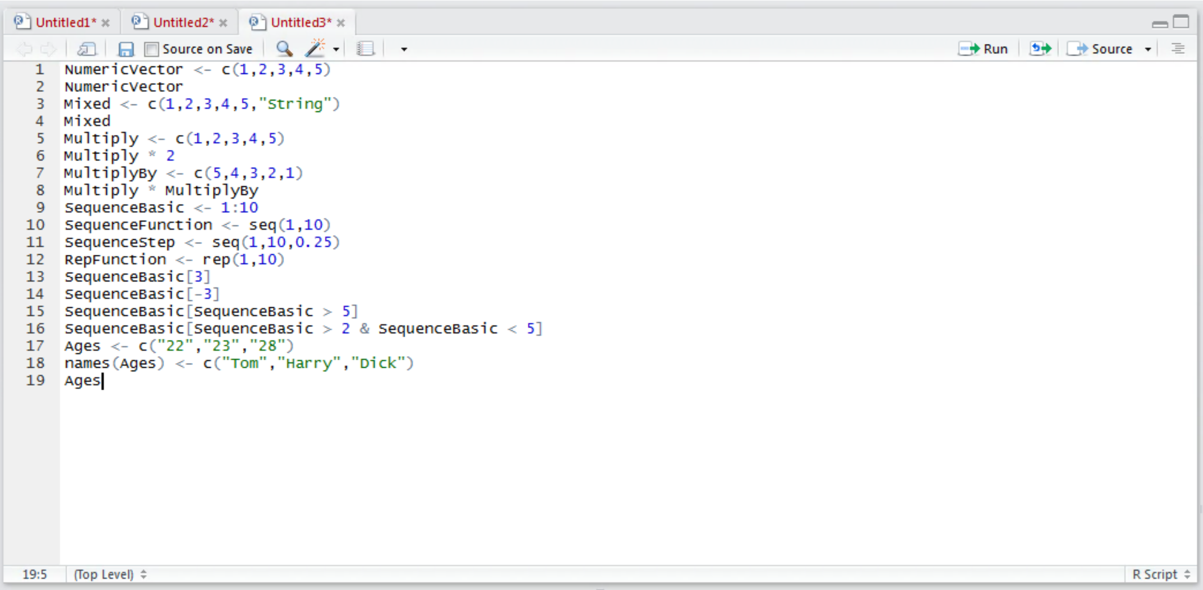
Run the line of script to console:

```
> Multiply <- c(1,2,3,4,5)
> Multiply * 2
[1]  2  4  6  8 10
> MultiplyBy <- c(5,4,3,2,1)
> Multiply * MultiplyBy
[1] 5 8 9 8 5
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
> SequenceBasic[SequenceBasic > 5]
[1]  6  7  8  9 10
> SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
[1] 3 4
> Ages <- c("22","23","28")
> names(Ages) <- c("Tom","Harry","Dick")
>
```

It can be observed that the Vector in the environment pane is now marked as being a ‘Named’ vector:



Outputting the Vector to console, type:



Run the line of script to console:

It can be observed that the vector more closely resembles the row of a spreadsheet. The names function will be used more extensively when aggregating Vectors into a Matrix, for the time being however, it will be used to allow for the selection of just that individuals Age.

Procedure 7: Selecting and Filtering from a Character Vector.

Once a Vector has been named, attaching a label to each value, it can be selected using the [] square bracket structure. In this example, the age for Tom needs to be extracted by typing:

Ages["Tom"]

```r
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
```

Run the line of script to console:

```
> SequenceBasic <- 1:10
> SequenceFunction <- seq(1,10)
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
> SequenceBasic[SequenceBasic > 5]
[1]  6  7  8  9 10
> SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
[1] 3 4
> Ages <- c("22","23","28")
> names(Ages) <- c("Tom","Harry","Dick")
> Ages
  Tom Harry  Dick 
 "22"  "23"  "28" 
> Ages["Tom"]
 Tom 
"22" 
>
```

Tom's age is returned as 22, rather the value in the Vector carrying the label "Tom" is returned as 22.

To select more than one label, it is a matter of creating a Vector with the criteria then passing that Vector inside the [] square brackets. In this example, selecting Tom and Dick:

Ages[c("Tom","Dick")]

```
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
```

Run the line of script to console:

```
> SequenceStep <- seq(1,10,0.25)
> RepFunction <- rep(1,10)
> SequenceBasic[3]
[1] 3
> SequenceBasic[-3]
[1]  1  2  4  5  6  7  8  9 10
> SequenceBasic[SequenceBasic > 5]
[1]  6  7  8  9 10
> SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
[1] 3 4
> Ages <- c("22","23","28")
> names(Ages) <- c("Tom","Harry","Dick")
> Ages
  Tom Harry  Dick 
 "22"  "23"  "28" 
> Ages["Tom"]
 Tom 
"22" 
> Ages[c("Tom","Dick")]
 Tom Dick 
"22" "28" 
> 
```

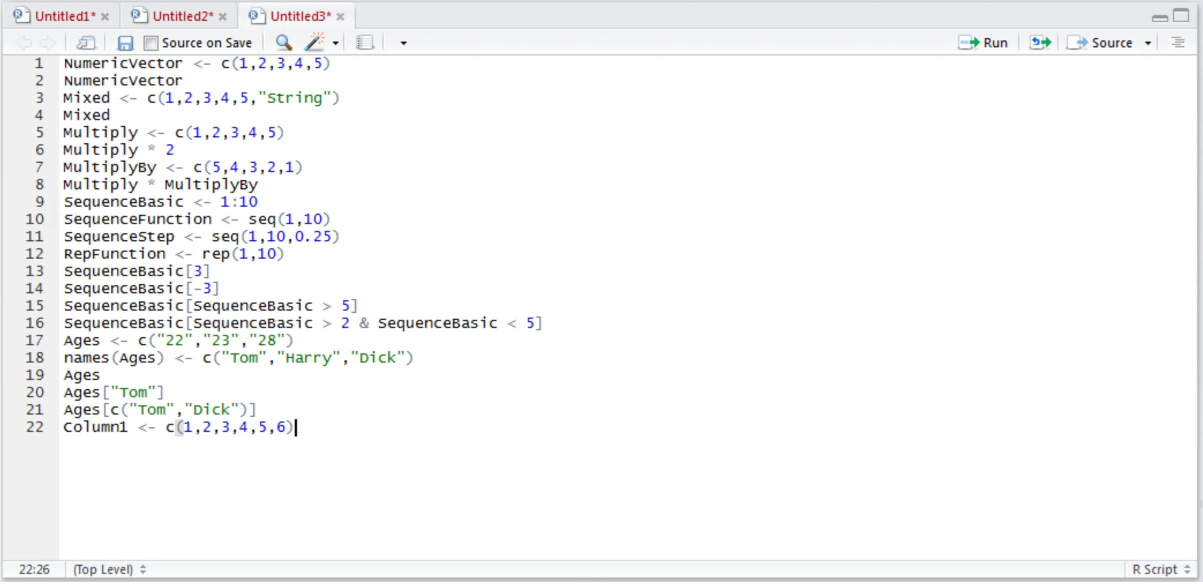
Procedure 8: Combine Vectors to make a Matrix with cbind.

A vector could be viewed as a column in an Excel spreadsheet. It folows that if there are several vectors, they would need to be brought together to create a similar structure. One structure that closely resembles a spreadsheet, working with the assumption that the contents of that spreadsheet is all the same data type, is a matrix.

To assume that every vector is to be a column in the matrix, the cbind() function is used to bring those columns togeter into this data structure.

To start, create three vectors of the same length:

Column1 <- c(1,2,3,4,5,6)

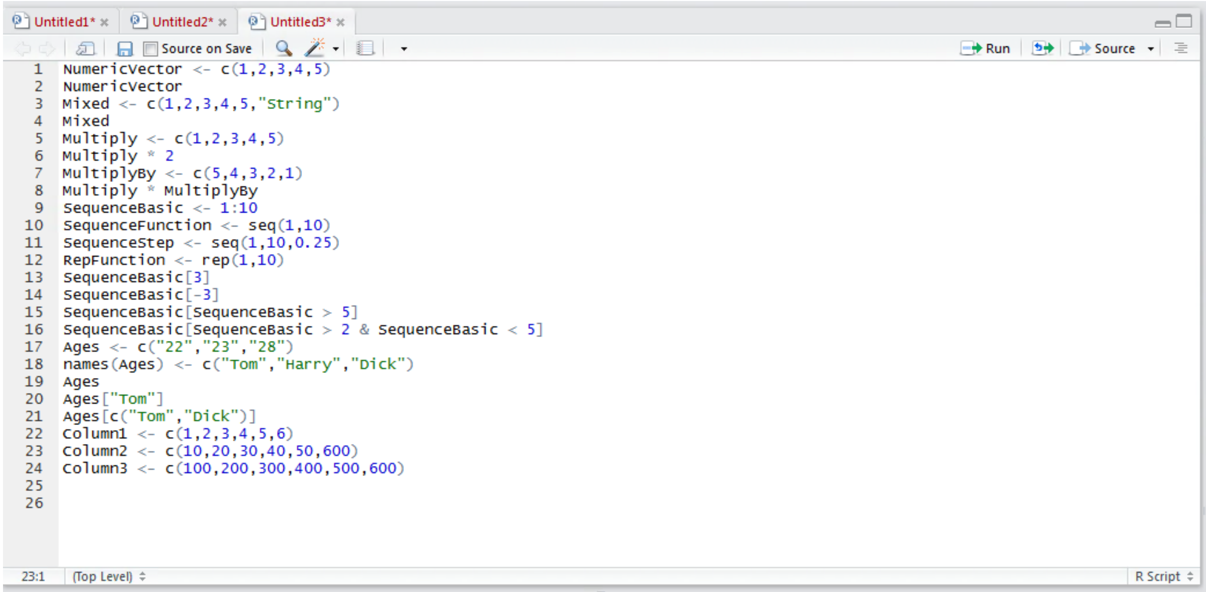


Run the line of script to the console:

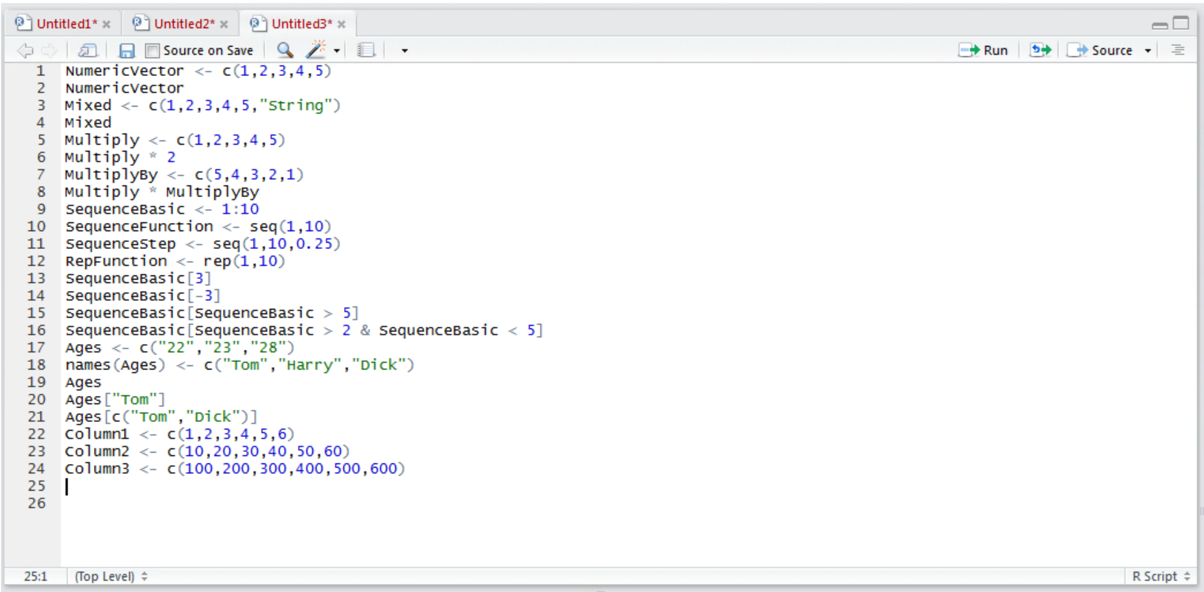
Repeat for two new columns, creating a script block:

Column2 <- c(10,20,30,40,50,60)

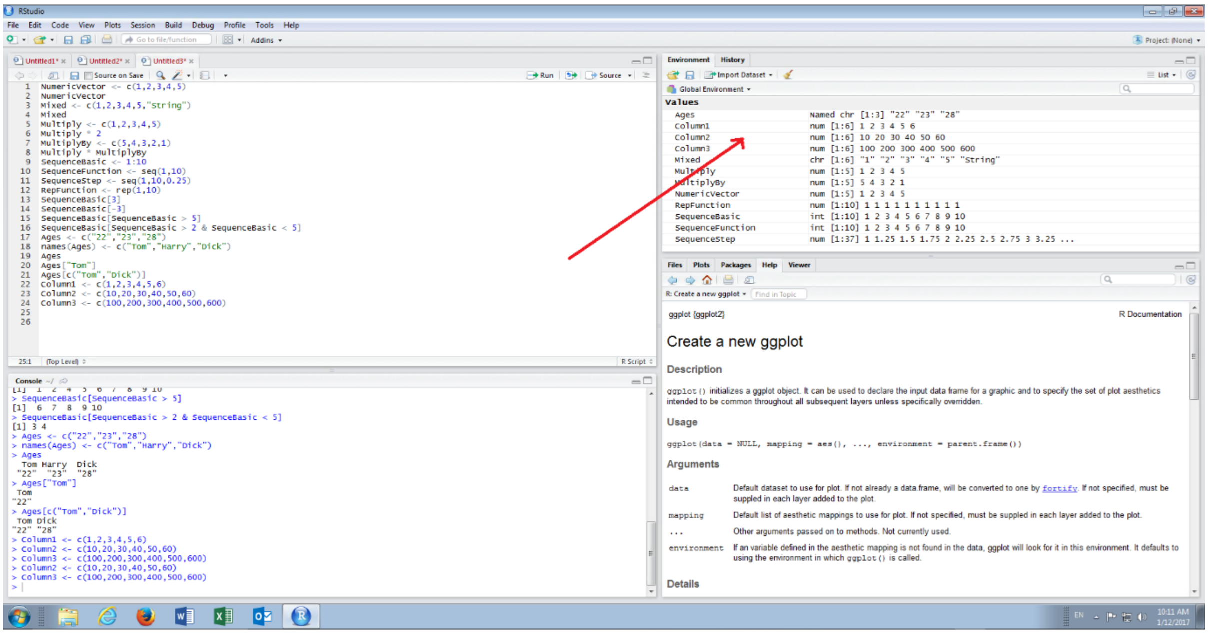
Column3 <- c(100,200,300,400,500,600)



Run each new line of script to console. It is important to note that each line of scrpt will have to be run to the console individually by navigating to the end of the line, clicking the Run button (or CTRL+Enter) and repeating a click of the Run button upon the cursor being moved to the next line. Hitherto this procedure of running more lines to console will be refered to as running the script block to console.

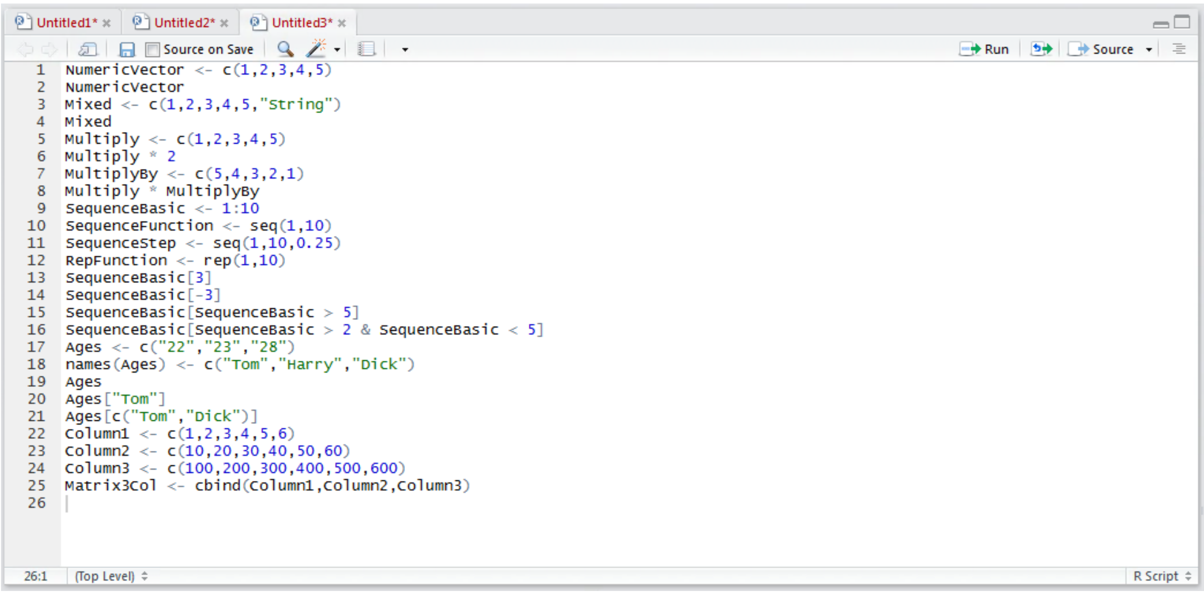


It can be observed that there are now three Vectors, columns, in the environment pane:

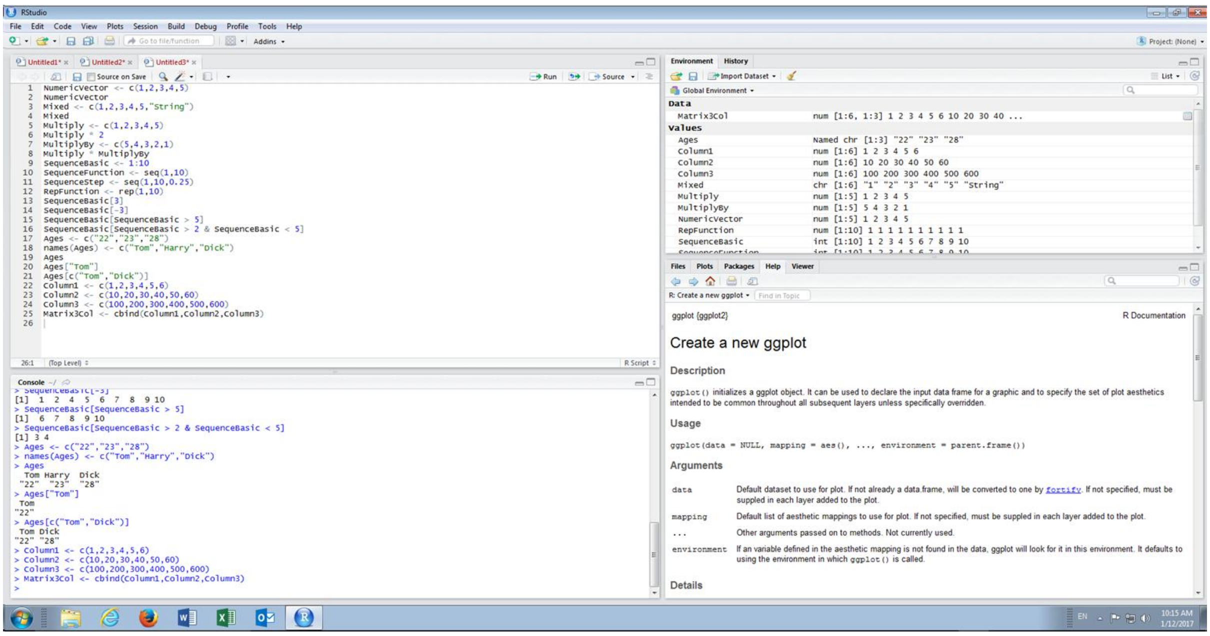


The task is to bring these columns together into a Matrix, rather bind these colums. The cbind() function is used to instruct this binding of columns. Type:

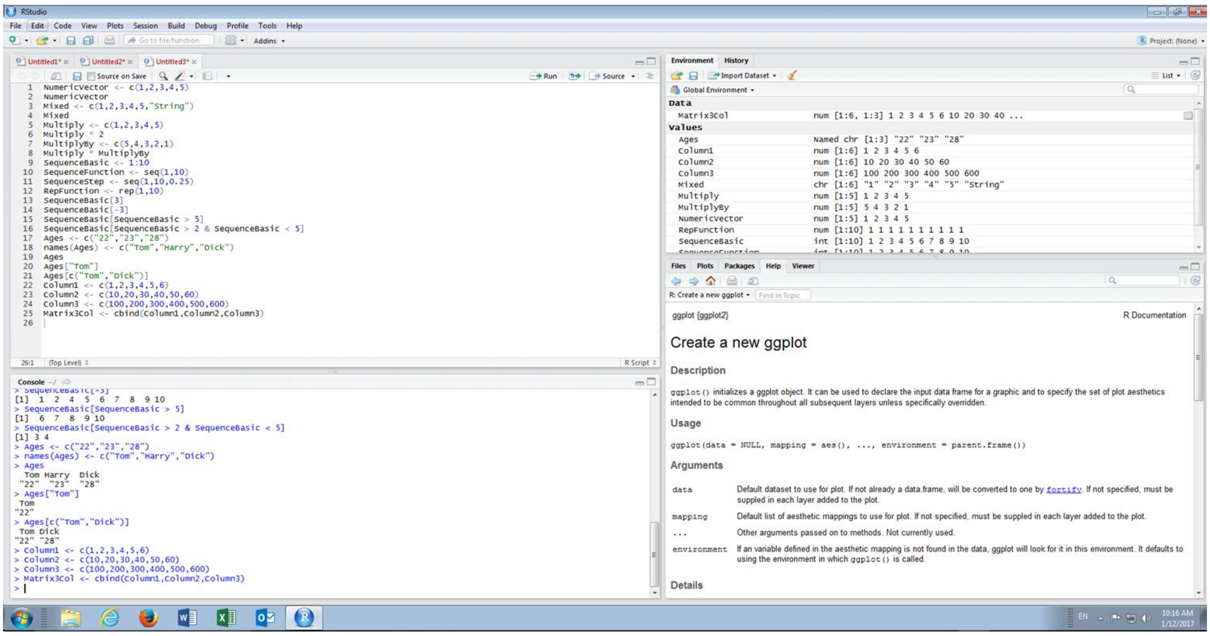
Matrix3Col <- cbind(Column1,Column2,Column3)



Run the line of script to console:



It can be seen that a new section in the environment pane has been created, titled Data:



Naturally the new matrix can be viewed by simply typing the Matrix name:

Matrix3Col

```r
NumericVector <- c(1,2,3,4,5)
NumericVector
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
```

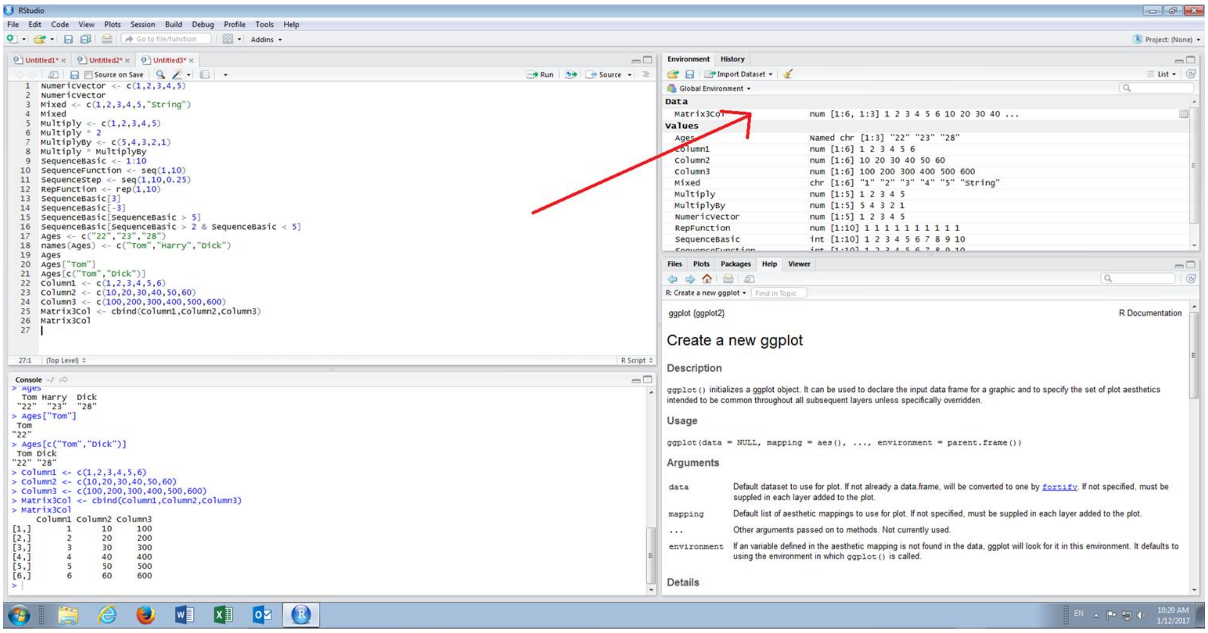
Run the line of script to console:

```
> Ages
  Tom Harry  Dick
 "22"  "23"  "28"
> Ages["Tom"]
 Tom
"22"
> Ages[c("Tom","Dick")]
 Tom Dick
"22" "28"
> Column1 <- c(1,2,3,4,5,6)
> Column2 <- c(10,20,30,40,50,60)
> Column3 <- c(100,200,300,400,500,600)
> Matrix3Col <- cbind(Column1,Column2,Column3)
> Matrix3Col
     Column1 Column2 Column3
[1,]       1      10     100
[2,]       2      20     200
[3,]       3      30     300
[4,]       4      40     400
[5,]       5      50     500
[6,]       6      60     600
>
```

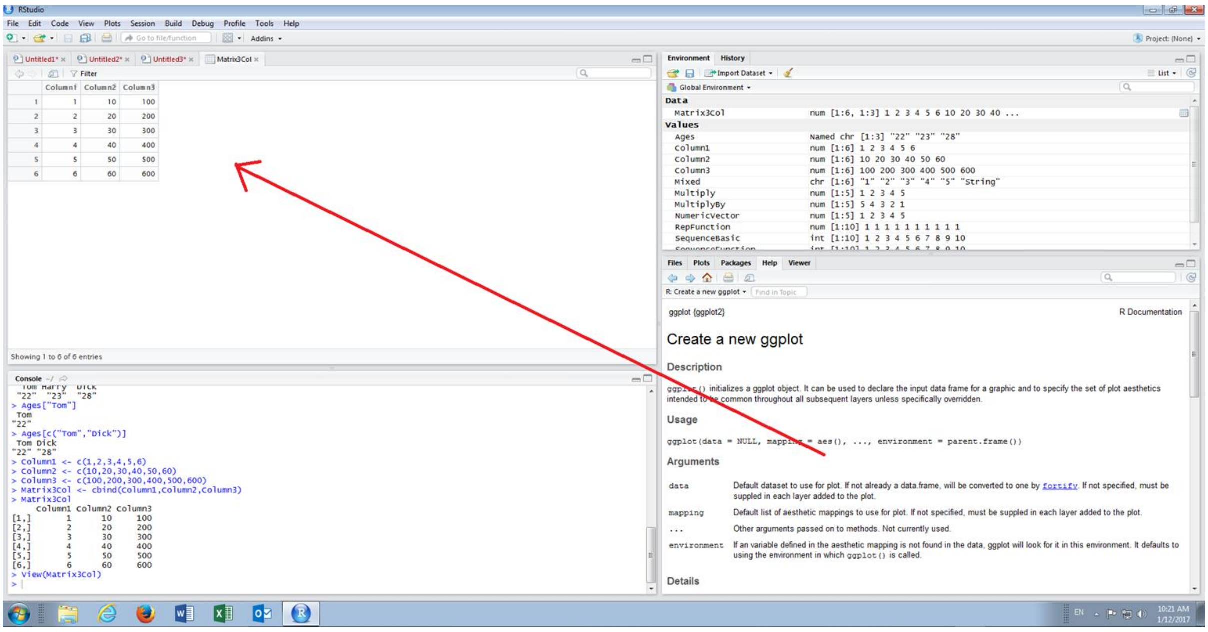
Procedure 9: Viewing a Matrix.

It can be observed that the matrix created in procedure 26 has been written out to the console. It was noted that there is a new section titled data in the environment pane, under which the matrix is displayed.

To expand the data into a tabbed grid, simply click with the mouse on the Matrix3Col under the data section of the environment pane:

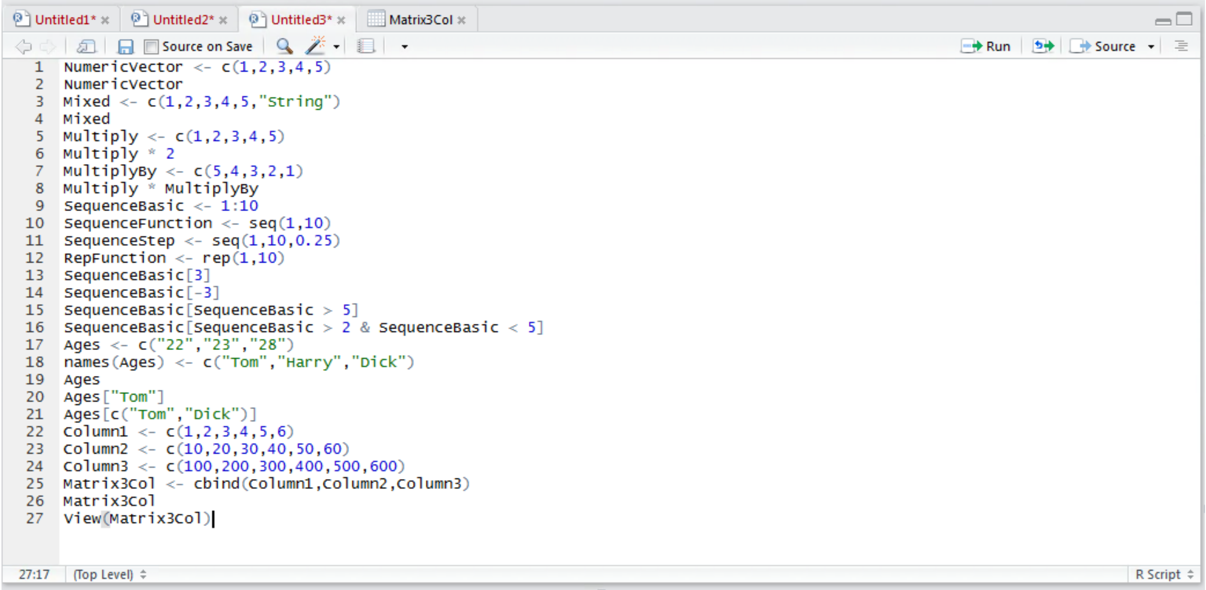


The tabbed grid will explode:

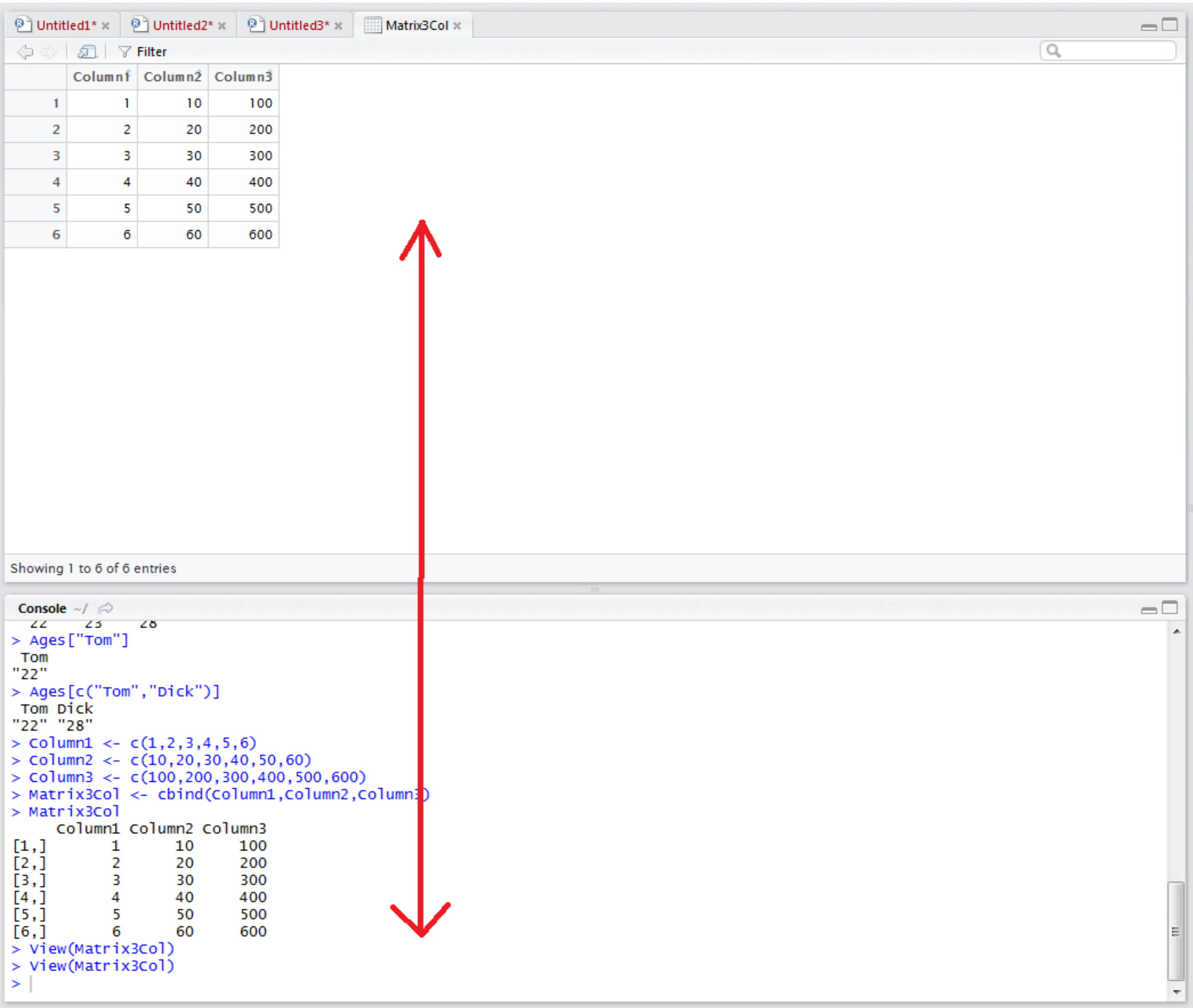


Note also that on clicking the matrix in the environment pane, that a script command has actually sent to the console. As such vieweing data in this manner can be invoked via a line of script. Using the script editor type:

View(Matrix3Col)



Run the line of script to the console:



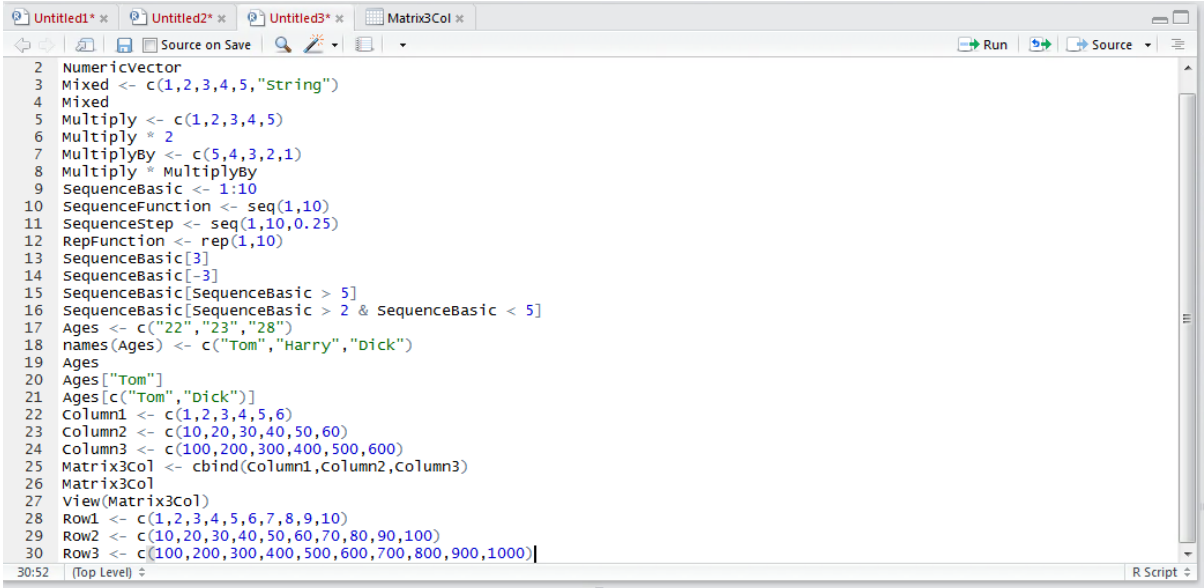
Procedure 10: Combine Vectors to make a Matrix with rbind.

Whereas procedure 26 brought vectors together as columns, rbind() can bring vectors together as rows. Start by creating two vectors in a script block:

Row1 <- c(1,2,3,4,5,6,7,8,9,10)

Row2 <- c(10,20,30,40,50,60,70,80,90,100)

Row3 <- c(100,200,300,400,500,600,700,800,900,1000)



Run the script block to console:

To bind the vectors as rows use the rbind() function:

Matrix3Row <- c(Row1,Row2,Row3)

```
Mixed <- c(1,2,3,4,5,"String")
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
```

Run the line of script to console:

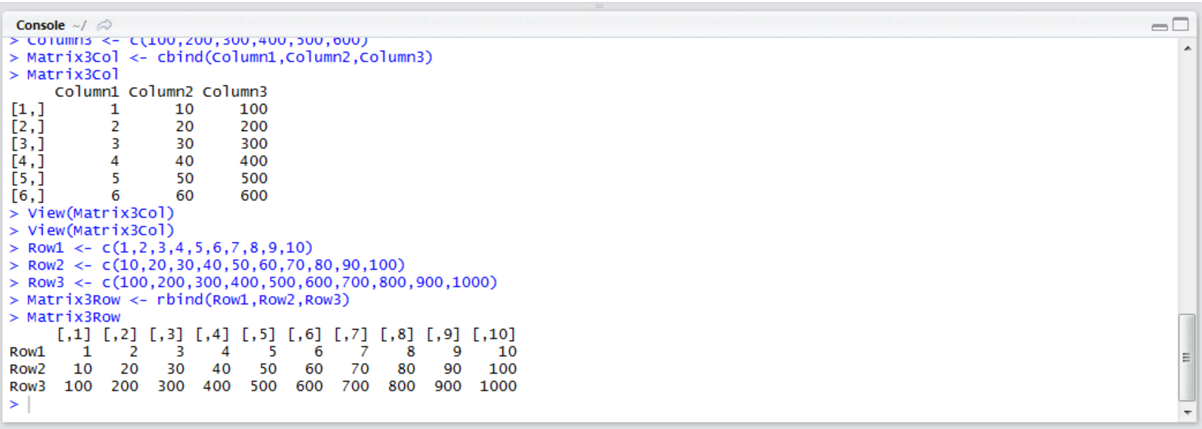
```
> Ages[c("Tom","Dick")]
 Tom Dick
"22" "28"
> Column1 <- c(1,2,3,4,5,6)
> Column2 <- c(10,20,30,40,50,60)
> Column3 <- c(100,200,300,400,500,600)
> Matrix3Col <- cbind(Column1,Column2,Column3)
> Matrix3Col
     Column1 Column2 Column3
[1,]       1      10     100
[2,]       2      20     200
[3,]       3      30     300
[4,]       4      40     400
[5,]       5      50     500
[6,]       6      60     600
> View(Matrix3Col)
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
>
```

The matrix can be viewed by typing:

Matrix3Row

```
Mixed
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
```

Run the line of script to console:



```
Console ~/
> Column3 <- c(100,200,300,400,500,600)
> Matrix3Col <- cbind(Column1,Column2,Column3)
> Matrix3Col
     Column1 Column2 Column3
[1,]       1      10     100
[2,]       2      20     200
[3,]       3      30     300
[4,]       4      40     400
[5,]       5      50     500
[6,]       6      60     600
> View(Matrix3Col)
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> 
```

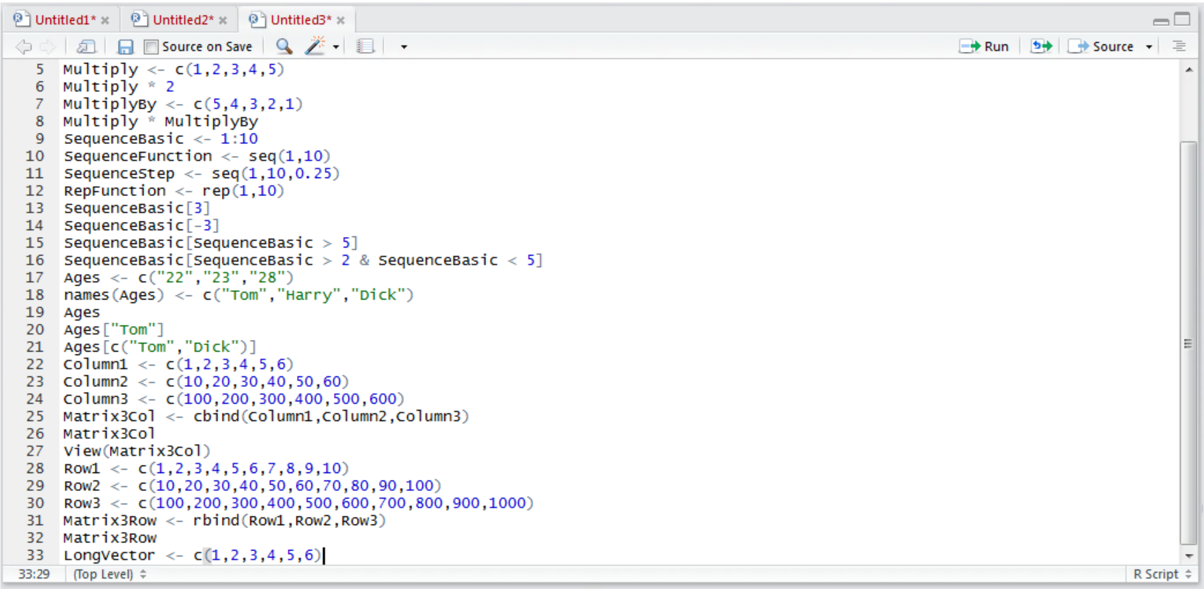
Procedure 11: Create a Matrix of defined size with a Vector.

Procedure 26 and 28 showed how to create a matrix using an intuitive method to bind vectors into columns and rows, comparing this to an Excel spreadsheet. It is possible to create a matrix with a given specification then fill that specification with a single vector which overspills the dimensions.

The matrix() function is intended to take a single vector as an argument coupled with the dimensions (i.e. the number of rows and columns). The single vector fills up this matrix by moving through each entry, downwards, in each column repeating the vector, should that vector not be long enough to fill up the matrix.

Start by creating a vector of six values by typing:

LongVector <- c(1,2,3,4,5,6)



```
Untitled1* Untitled2* Untitled3*
Source on Save    Run    Source
Multiply <- c(1,2,3,4,5)
Multiply * 2
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
33:29   (Top Level)   R Script
```

Run the line of script to console:

```
> Column3 <- c(100,200,300,400,500,600)
> Matrix3Col <- cbind(Column1,Column2,Column3)
> Matrix3Col
     Column1 Column2 Column3
[1,]       1      10     100
[2,]       2      20     200
[3,]       3      30     300
[4,]       4      40     400
[5,]       5      50     500
[6,]       6      60     600
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
>
```

Bearing in mind that the matrix will fill up column wise, make a matrix that is only three rows deep, while being four columns wide (i.e. nrow=3,ncol=4):

matrix(LongVector,nrow = 3,ncol = 4)

```
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
```

Run the line of script to console:

```
> Matrix3Col <- cbind(Column1,Column2,Column3)
> Matrix3Col
     Column1 Column2 Column3
[1,]       1      10     100
[2,]       2      20     200
[3,]       3      30     300
[4,]       4      40     400
[5,]       5      50     500
[6,]       6      60     600
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
>
```

To view the matrix and specifically how the LongVector overlaid this matrix type:

OverspillMatrix

```r
MultiplyBy <- c(5,4,3,2,1)
Multiply * MultiplyBy
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
```

Run the line of script to console:

```
[3,]        3     30     300
[4,]        4     40     400
[5,]        5     50     500
[6,]        6     60     600
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
>
```

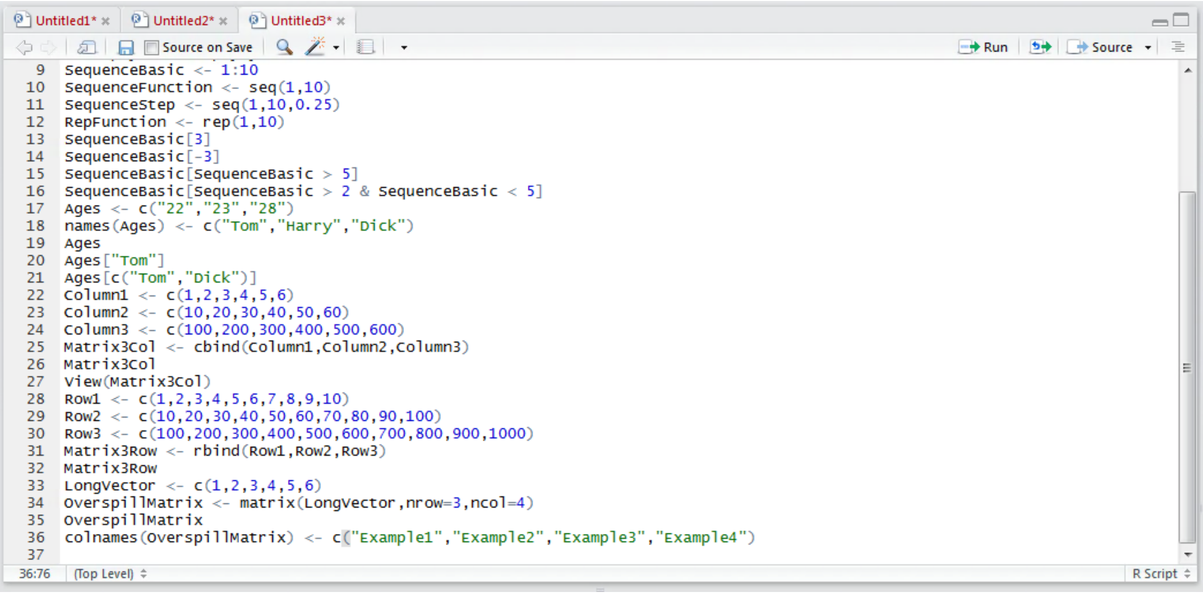
It can be seen that in moving column wise, when the vector runs out, it starts again until the matrix has been filled as per the dimensions.

Procedure 12: Labelling a Matrix.

As seen in procedure 24 it is helpful for reference to label a Vector. It is possible also to label the rows and the columns of a matrix in a similar fashion using the rownames() and colnames() function.

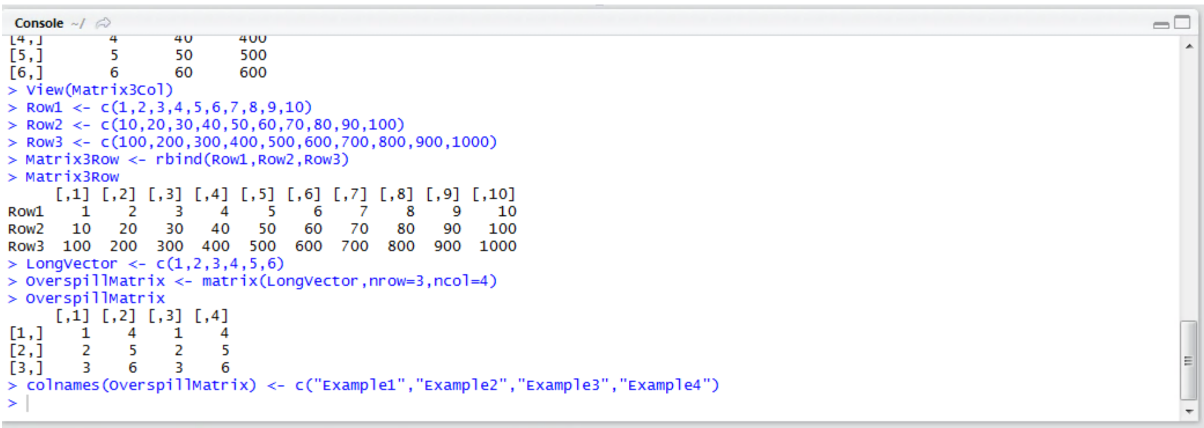
To set column names assign a Vector to the colnames() function, where the colnames() function accepts the matrix as its argument:

colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")



```
SequenceBasic <- 1:10
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
```

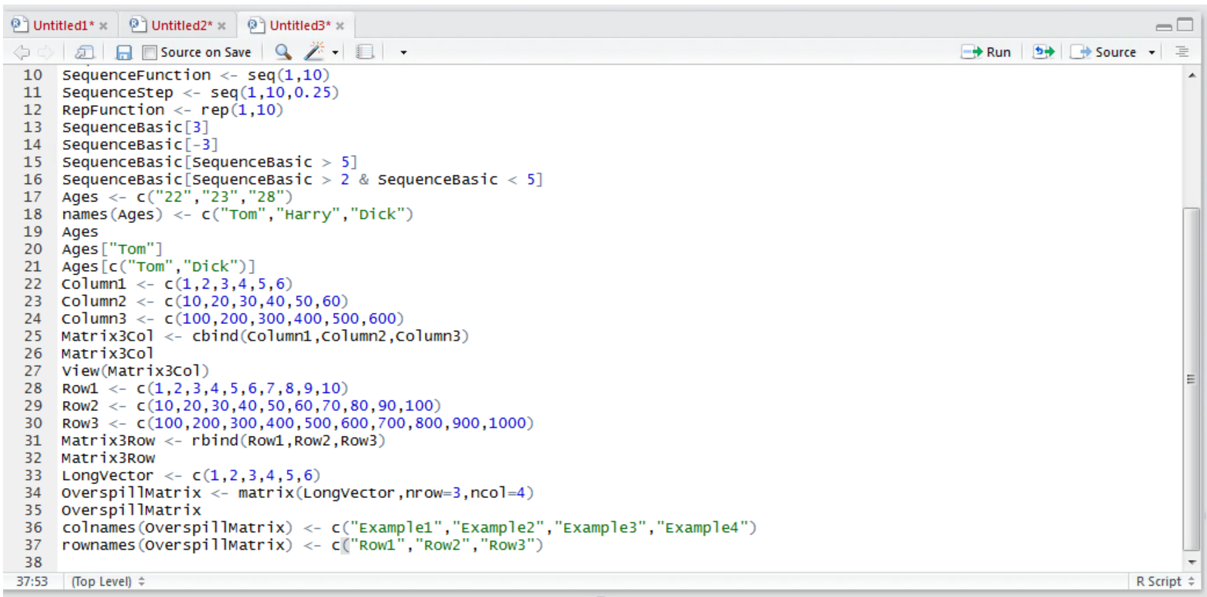
Run the line of script to console:



```
[5,]       5      50     500
[6,]       6      60     600
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
>
```

The rownames() function has a similar signature and takes a Vector of row names:

rownames(OverspillMatrix) <- c("Row1","Row2","Row3")



```
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
```

Run the line of script to console:

```
Console ~/
[5,]    5    50    500
[6,]    6    60    600
> View(Matrix3Col)
> Row1 <- c(1,2,3,4,5,6,7,8,9,10)
> Row2 <- c(10,20,30,40,50,60,70,80,90,100)
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> 
```

The matrix is now labelled in both directions and can be inspected by typing:

OverspillMatrix

```
Untitled1* × Untitled2* × Untitled3* ×
Source on Save                                   Run    Source
SequenceFunction <- seq(1,10)
SequenceStep <- seq(1,10,0.25)
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
38:16  (Top Level) ÷                                                R Script ÷
```

Run the line of script to console:

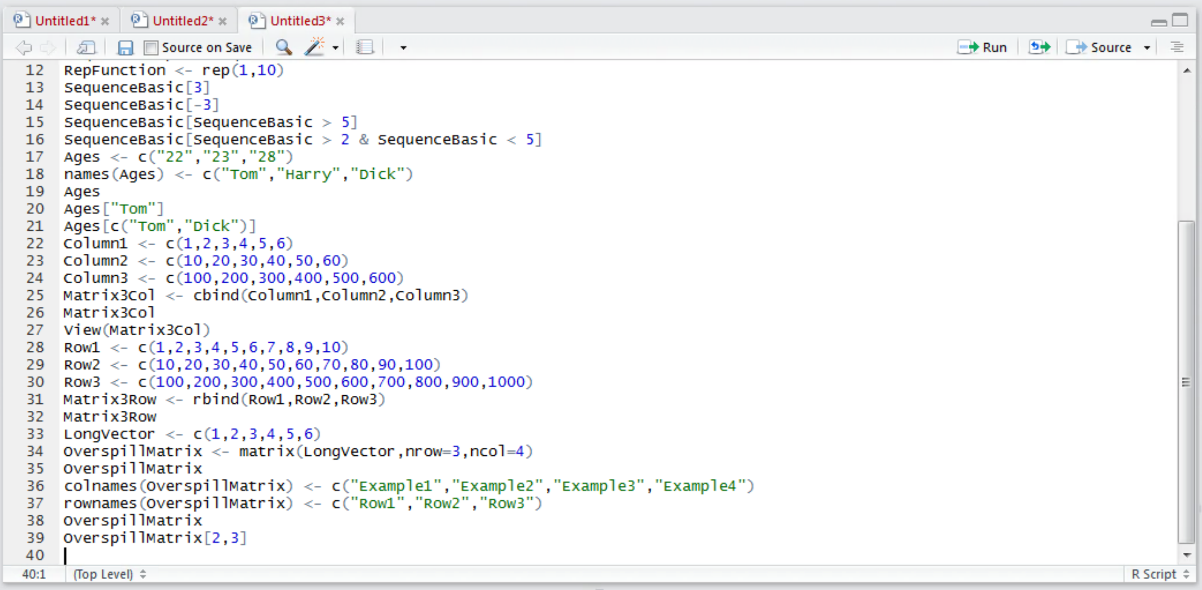
```
Console ~/
> Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
> Matrix3Row <- rbind(Row1,Row2,Row3)
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> OverspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> 
```

Procedure 13: Selecting from a Matrix.

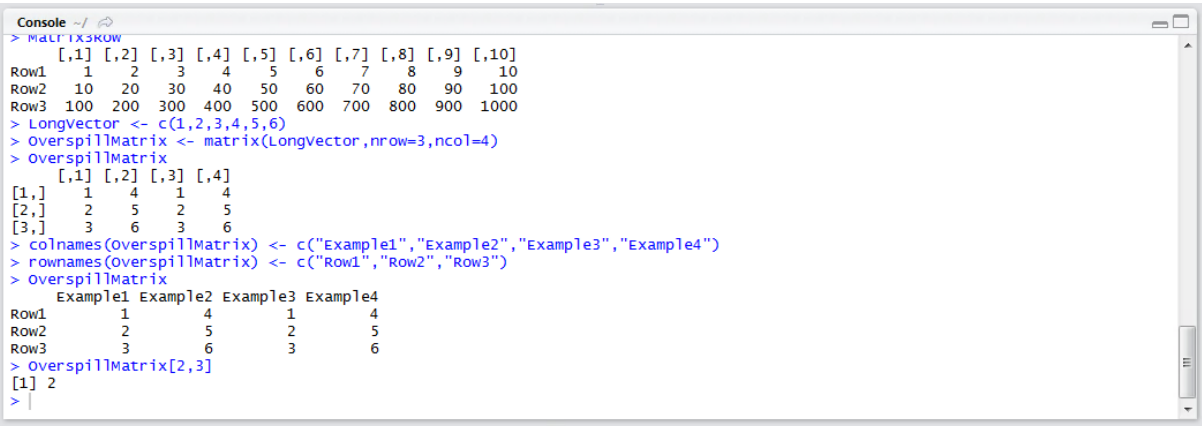
As a matrix is made up of vectors, it is logical to expect it to bear some resemblance in the way selection from a matrix takes place. All subscripting types that are described in procedures 24 and 25, are available except for a separate dimension is specified inside the [] square brackets, as separate arguments. The first argument inside the square brackets relates to the row, the next the column.

To obtain the value in a given position of a matrix, in this case two down, three across, type:

OverspillMatrix[2,3]



Run the line of script to console:



It can be seen that the value 2 has been returned which corresponds to the position specified:

```
Console ~/
> Matrix3Row
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> OverspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> OverspillMatrix[2,3]
[1] 2
>
```

Procedure 14: Creating a Factor from a Vector.

The factor() function turns a Vector containing character fields into a special structure for categorical variables. Categorical variables are treated differently in data analysis as conceptually they are pivoted to columns in their own right.

Assume that a Vector of customer genders exists:

Gender <- c("Male","Female","Female","Male")

```
Untitled1* × Untitled2* × Untitled3* ×
Source on Save                                Run    Source
RepFunction <- rep(1,10)
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
40:45  (Top Level) ÷                                               R Script ÷
```

Run the line of script to console:

```
Console ~/
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> OverspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> OverspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
>
```

A standard vector has been created. To transform this Vector into a Factor, simply pass the Gender Vector as an argument to the factor() function by typing:

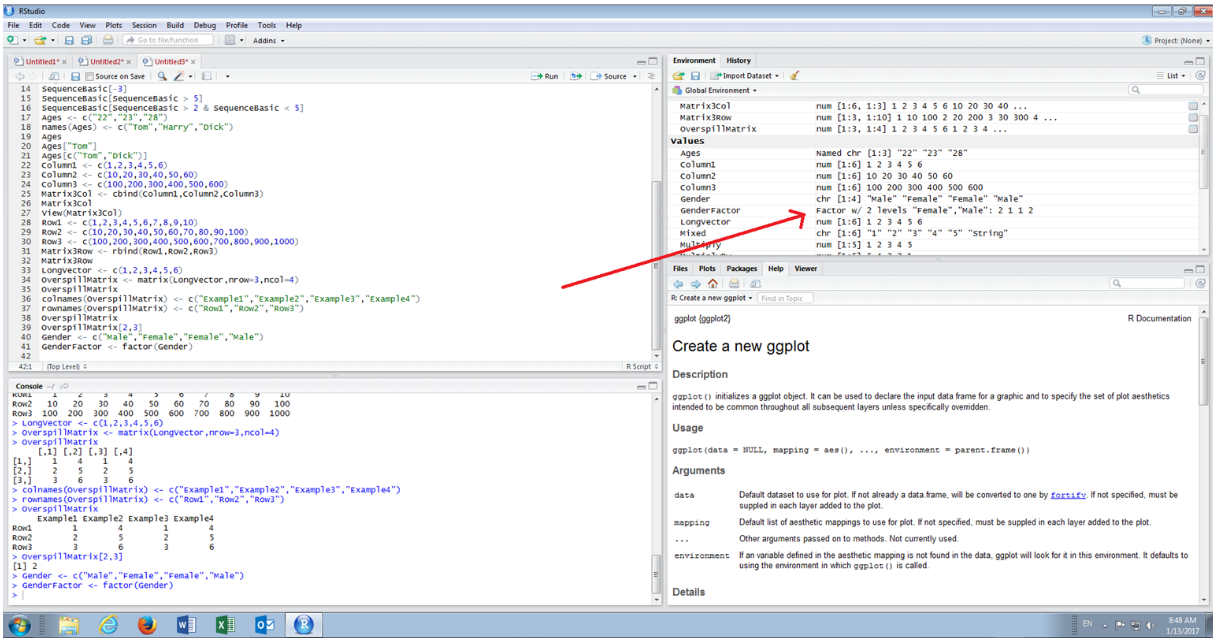
GenderFactor <- factor(gender)

```
SequenceBasic[3]
SequenceBasic[-3]
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
```

Run the line of script to console:

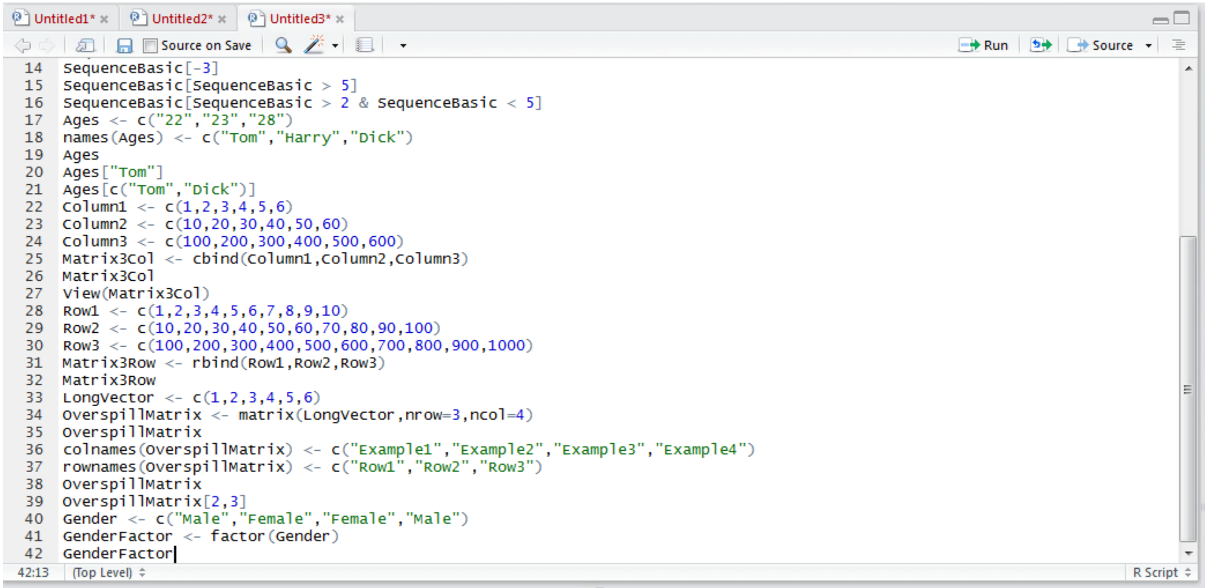
```
Row1    1    2    3    4    5    6    7    8    9   10
Row2   10   20   30   40   50   60   70   80   90  100
Row3  100  200  300  400  500  600  700  800  900 1000
> LongVector <- c(1,2,3,4,5,6)
> OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> OverspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> OverspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> OverspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
> GenderFactor <- factor(Gender)
>
```

It can be observed that the Factor is now available in the environment pane:



To view the factor in the console type:

GenderFactor



Run the line of script to console:

```
> LongVector <- c(1,2,3,4,5,6)
> overspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
> overspillMatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(overspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(overspillMatrix) <- c("Row1","Row2","Row3")
> overspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> overspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
> GenderFactor <- factor(Gender)
> GenderFactor
[1] Male   Female Female Male  
Levels: Female Male
> 
```

Closer inspection shows that despite there being a vector of the strings Male and Female duplicated, the Factor has correctly identified there to be two levels of Male and Female. This procedure is an example of the levels being inferred. Catagorical data will not be treated nativily in the predictive analytics tools as follows.

Procedure 15: Creating a Factor from a Vector with Levels and Ordering.

Some categorical data does also have a precedence whereby each of the categorical variables is somehow elevated from the previous one, while not necessarily being distributed in a statistical fashion. A good example would be temperature. Start by creating a Vector called Temps:

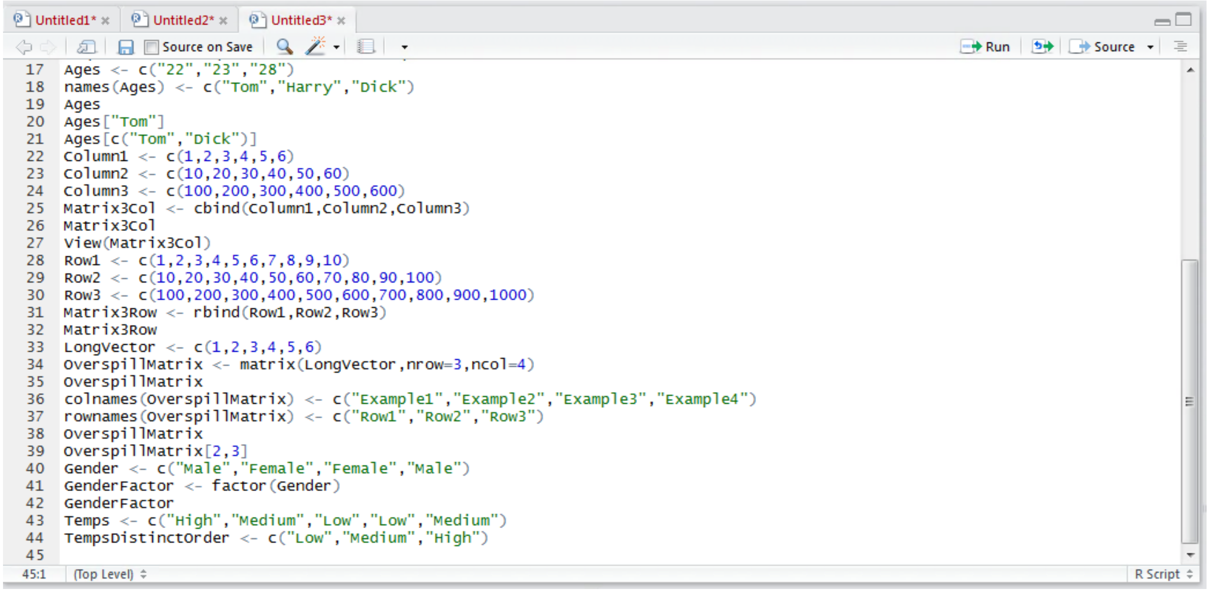
Temps <- c("High","Medium","Low","Low","Medium")

```
SequenceBasic[SequenceBasic > 5]
SequenceBasic[SequenceBasic > 2 & SequenceBasic < 5]
Ages <- c("22","23","28")
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
overspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
overspillMatrix
colnames(overspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(overspillMatrix) <- c("Row1","Row2","Row3")
overspillMatrix
overspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
```

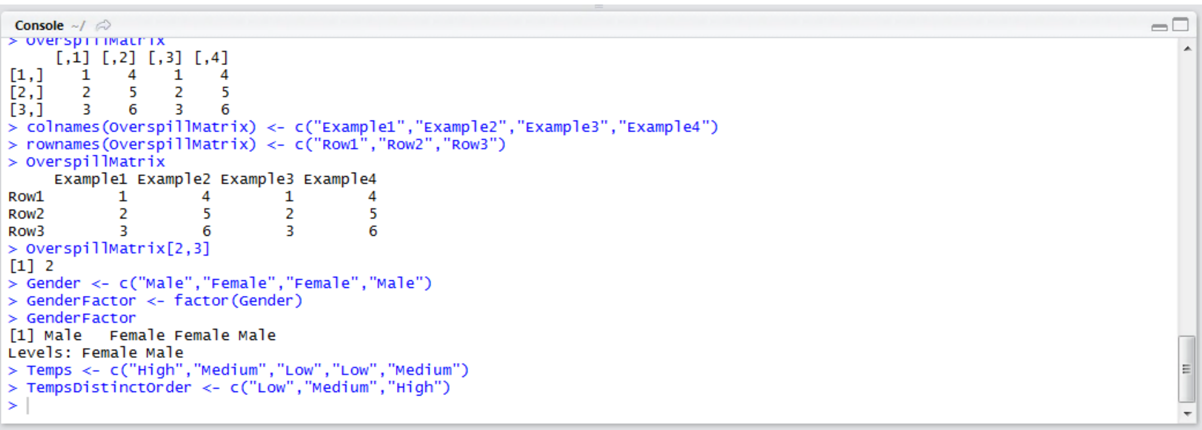
Run the line of script to console:

Create a similar Vector, this time with the distinct values in the order of precidence:

TempsDistinctOrder <- c("Low","Medium","High")



Run the line of script to console:



Create the factor by bringing the two newly created Vectors trogether and specfying that ordering is to be observed:

TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)

```r
names(Ages) <- c("Tom","Harry","Dick")
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)

46:1   (Top Level)                                          R Script
```

Run the line of script to console:

```
Console ~/
     [,1] [,2] [,3] [,4]
[1,]    1    4    1    4
[2,]    2    5    2    5
[3,]    3    6    3    6
> colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
> OverspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> OverspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
> GenderFactor <- factor(Gender)
> GenderFactor
[1] Male   Female Female Male
Levels: Female Male
> Temps <- c("High","Medium","Low","Low","Medium")
> TempsDistinctOrder <- c("Low","Medium","High")
> TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
> 
```

Write the Factor to console by typing:

TempsFactor

```r
Ages
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor

47:1   (Top Level)                                          R Script
```

Run the line of script to console:

```
Console ~/
[3,]    3    6    3    6
> colnames(overspillMatrix) <- c("Example1","Example2","Example3","Example4")
> rownames(overspillMatrix) <- c("Row1","Row2","Row3")
> overspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> overspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
> GenderFactor <- factor(Gender)
> GenderFactor
[1] Male   Female Female Male  
Levels: Female Male
> Temps <- c("High","Medium","Low","Low","Medium")
> TempsDistinctOrder <- c("Low","Medium","High")
> TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
> TempsFactor
[1] High   Medium Low    Low    Medium
Levels: Low < Medium < High
> 
```

It can be seen that the Factor levels now have < chevrons which denote the precidence. Low is less than Medium, Medium is less than High. Rather usefuly it is possible to use a logical test condition to perform a logical test for only those values in the factor that exceed a given level, for example type:

TempsFactor > "Low"

```
Untitled1* × Untitled2* × Untitled3* ×
Source on Save          Run    Source
Ages["Tom"]
Ages[c("Tom","Dick")]
Column1 <- c(1,2,3,4,5,6)
Column2 <- c(10,20,30,40,50,60)
Column3 <- c(100,200,300,400,500,600)
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
overspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
overspillMatrix
colnames(overspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(overspillMatrix) <- c("Row1","Row2","Row3")
overspillMatrix
overspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"

48:1   (Top Level)                                        R Script
```

Run the line of script to console:

```
Console ~/
> rownames(overspillMatrix) <- c("Row1","Row2","Row3")
> overspillMatrix
     Example1 Example2 Example3 Example4
Row1        1        4        1        4
Row2        2        5        2        5
Row3        3        6        3        6
> overspillMatrix[2,3]
[1] 2
> Gender <- c("Male","Female","Female","Male")
> GenderFactor <- factor(Gender)
> GenderFactor
[1] Male   Female Female Male  
Levels: Female Male
> Temps <- c("High","Medium","Low","Low","Medium")
> TempsDistinctOrder <- c("Low","Medium","High")
> TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
> TempsFactor
[1] High   Medium Low    Low    Medium
Levels: Low < Medium < High
> TempsFactor > "Low"
[1]  TRUE  TRUE FALSE FALSE  TRUE
> 
```

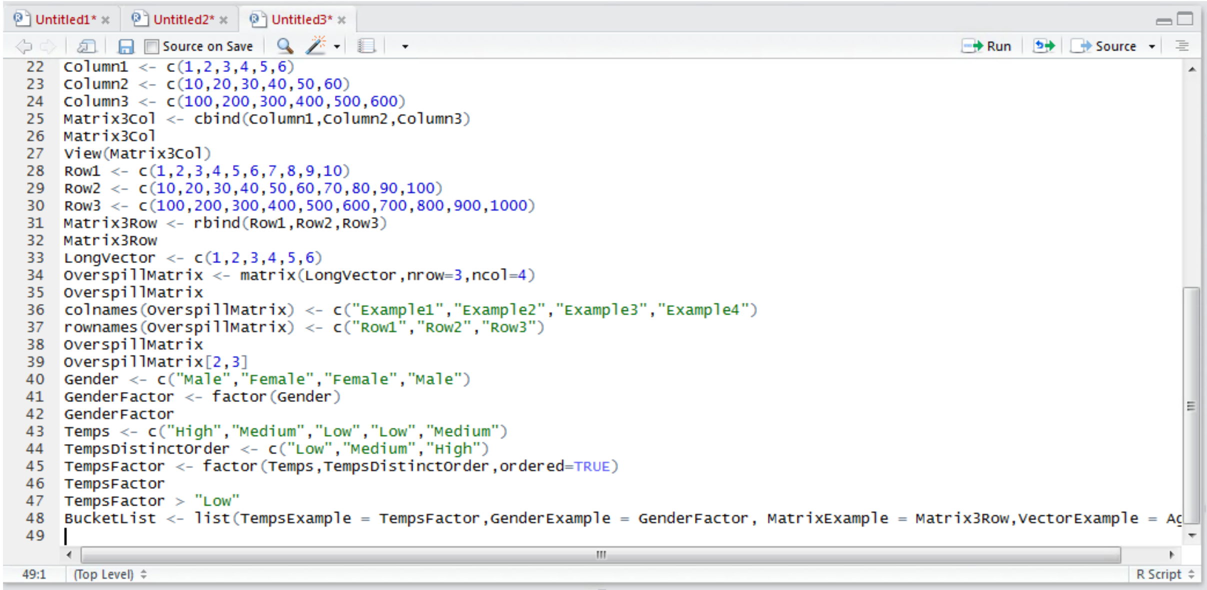
It can be seen that a Vector of logical operators has been returned that could further be used for selecting and subsetting.

Procedure 16: Creating a list with a variety of objects.

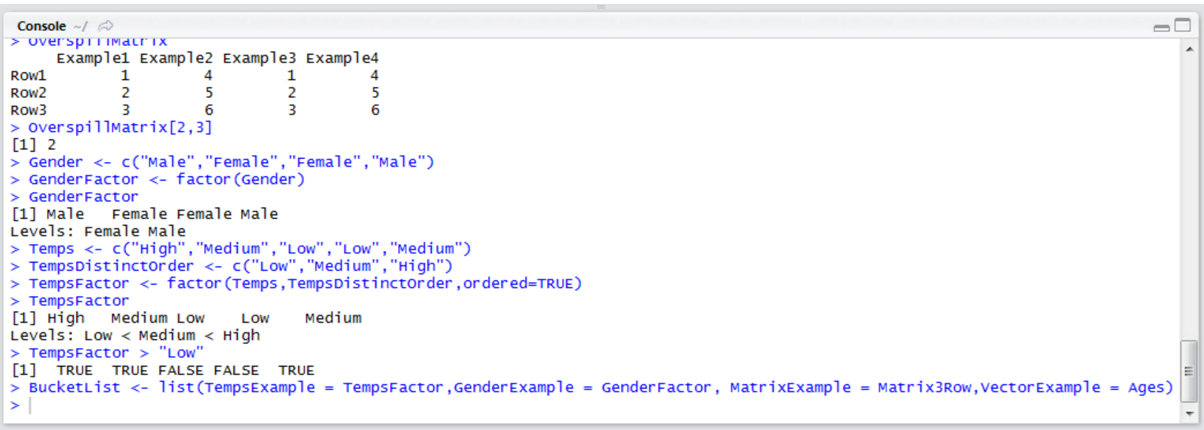
A list is very similar to a Vector excet it allows the storage of more than one type of object, whereas a Vector must be the same type. In the procedures preceeding, many objects have been created. A list can bring these objects together despite them being of radically different types.

The list() function, used to create lists, is very similar to that of the c() function except it has a broader ability to specify object names at creation. To create a list aggregating some objects created in the preceeding procedures:

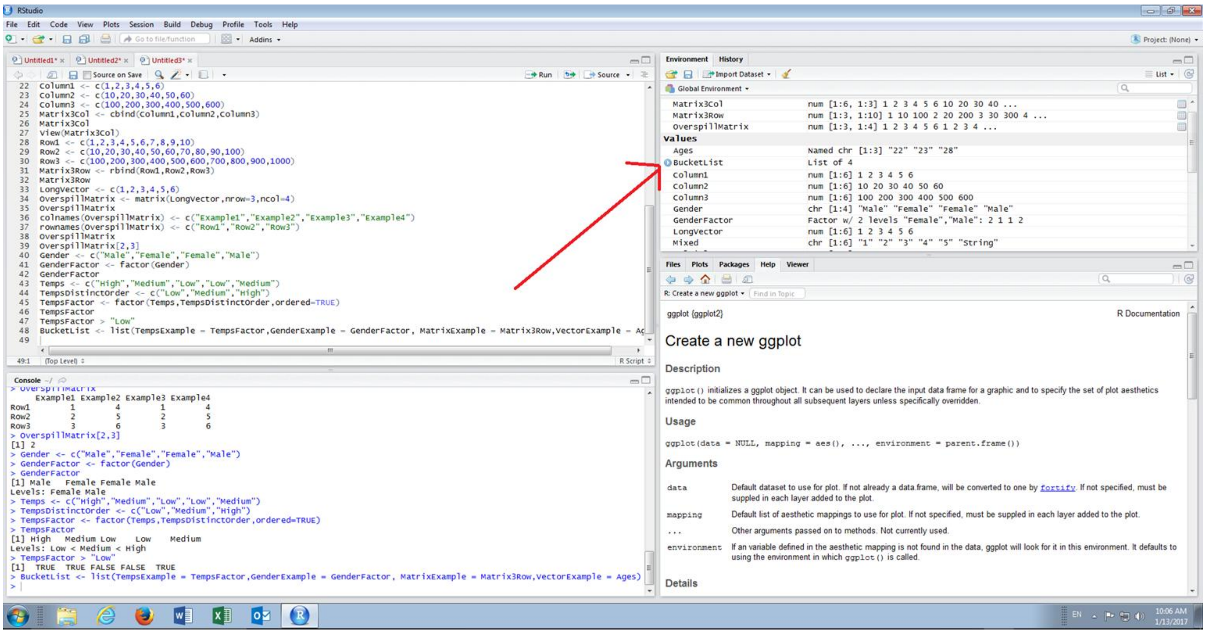
BucketList <- list(TempsExample = TempsFactors,GenderExample = GenderFactors,MatrixExample = Matrix3Row,VectorExample = Ages)



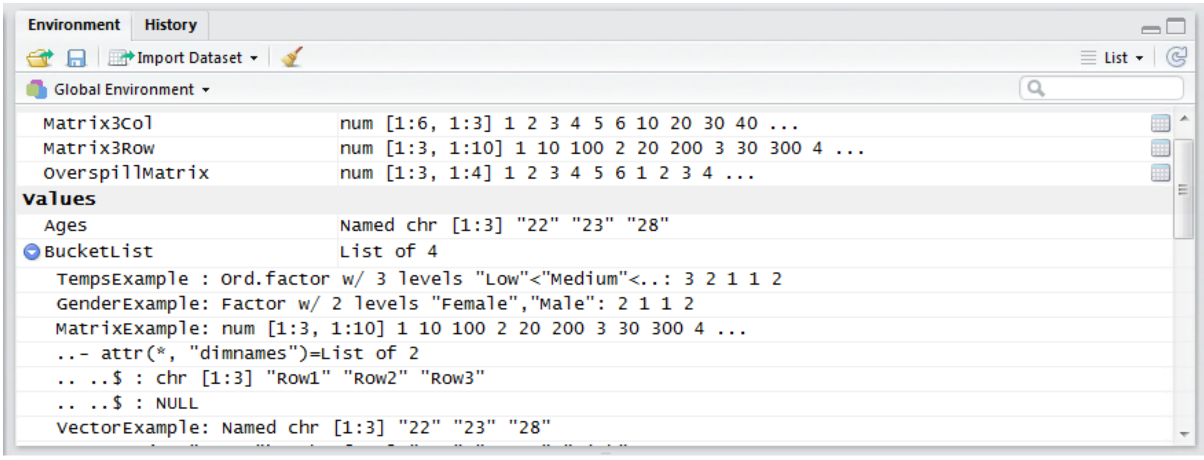
Run the line of script to console:



It can be seen that the list is now available in the environmennt pane:

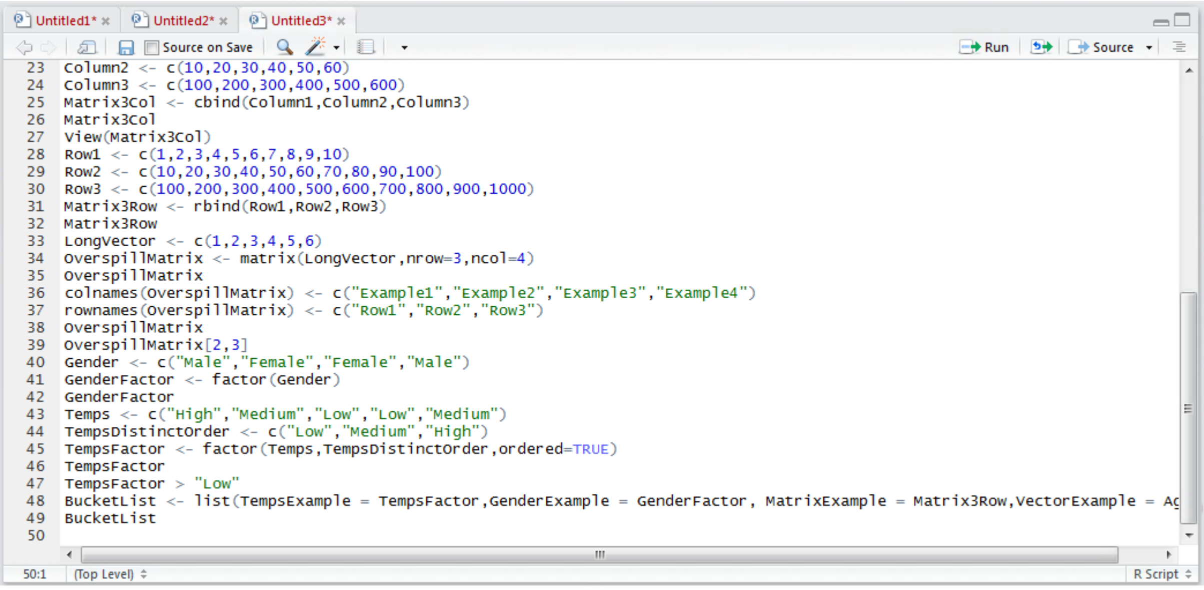


Specifically it is possible, by clicking on the play icon, to expand the list and inspect the objects inside the list in turn:



To write out the entier contents of the list to the console type:

BucketList



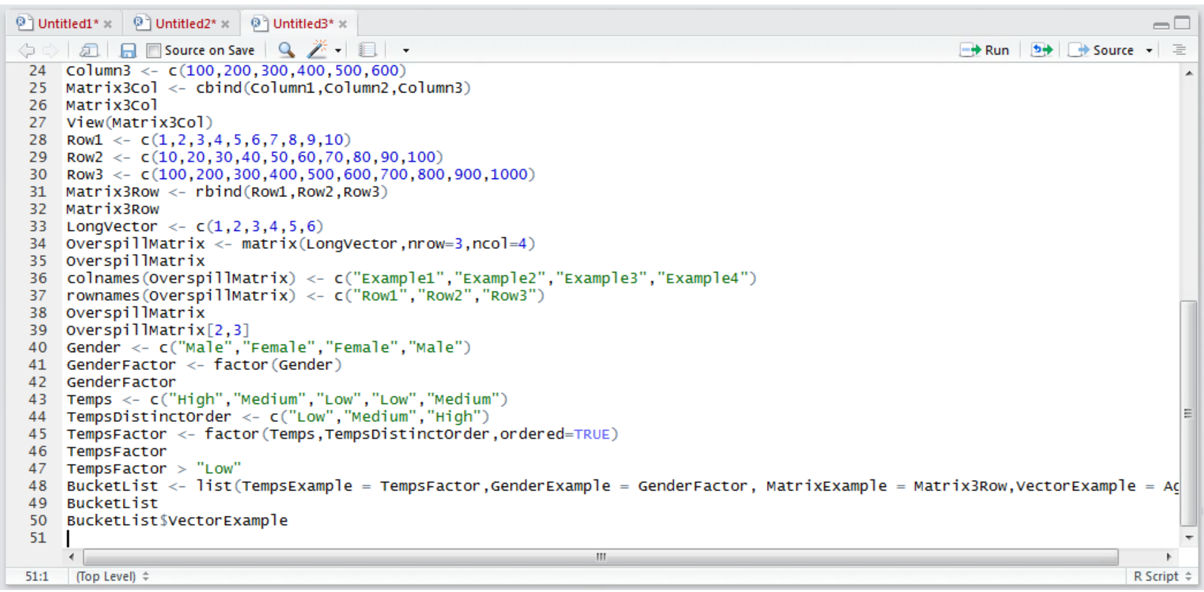
Run the line of script to console. It can be seen that each item of the list and its contents have been written out in turn.

Procedure 17: Subsetting and referencing objects with a name.

The most useful and common way to navigate a list is by referencing the entry in the list by name then subsetting the object therafter. The approach of referencing list objects by name, then subsetting therafter can serve to make a distinction between a list and a vector in day to day use.

The list created in procedure 34 has several objects with the names TempsExample, GenderExample,MatrixExample and VectorExample. Start by returning a vector object by name:

BucketList$VectorExample



Run the line of script to console:

JUBE

```
$TempsExample
[1] High   Medium Low    Low    Medium
Levels: Low < Medium < High

$GenderExample
[1] Male   Female Female Male
Levels: Female Male

$MatrixExample
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000

$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
>
```

It can be seen that the object stored under the name "VectorExample" is a labeled Vector. As this is a Vector, it is possible to further subset this using techniques outlined in procedure 25. For example, to return Tom's age from the Vector, type:

BucketList$VectorExample["Tom"]

```
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ag
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
```

Run the line of script to console:

```
$GenderExample
[1] Male   Female Female Male
Levels: Female Male

$MatrixExample
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000

$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
>
```

It can be observed that the vector was drawn from the list by name, then subset as is customary for a vector.

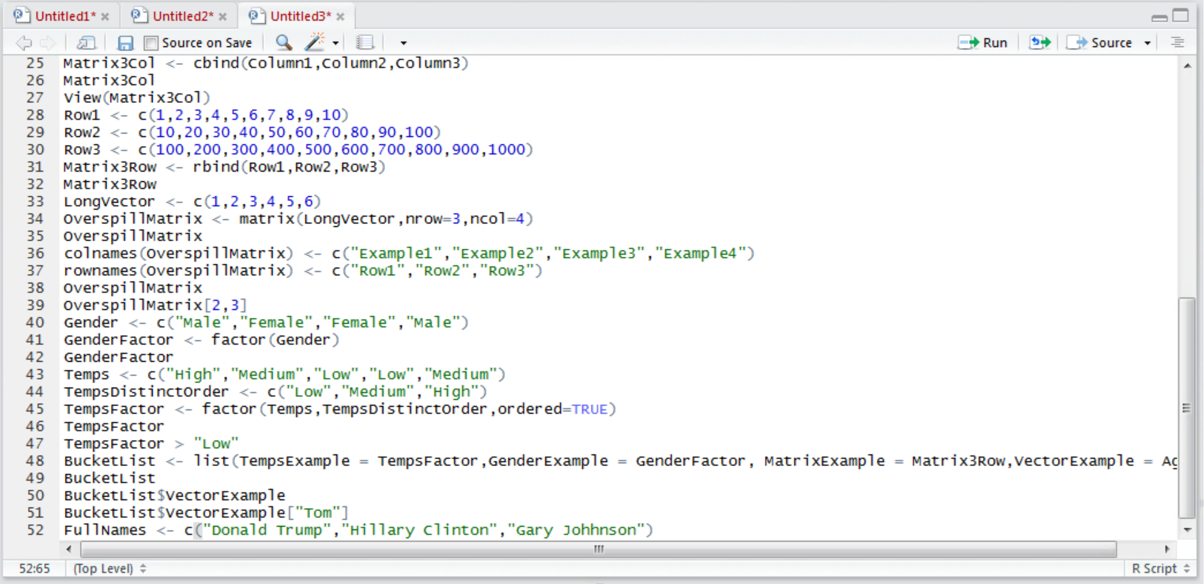
Procedure 18: Create a Data Frame from Vectors.

For the great majority of procedures that follow in this document the Data Frame is clearly demonstrated to be the most important and ubiquitos data structure. In its core a Data Frame is a list albiet with certain costraints. A data frame can only make use of Vectors and Factors and furthermore these objects need to be of EXACTLY the same length.

It can be helpful to thing of a Data Frame a being a hybrid of a Matrix and a List, with a great deal more usability than a Matrix. It is worth remembering that owing to the presence of Factors and Vectors, this is to say different object types, a matrix could not be used in all pracitcailty.

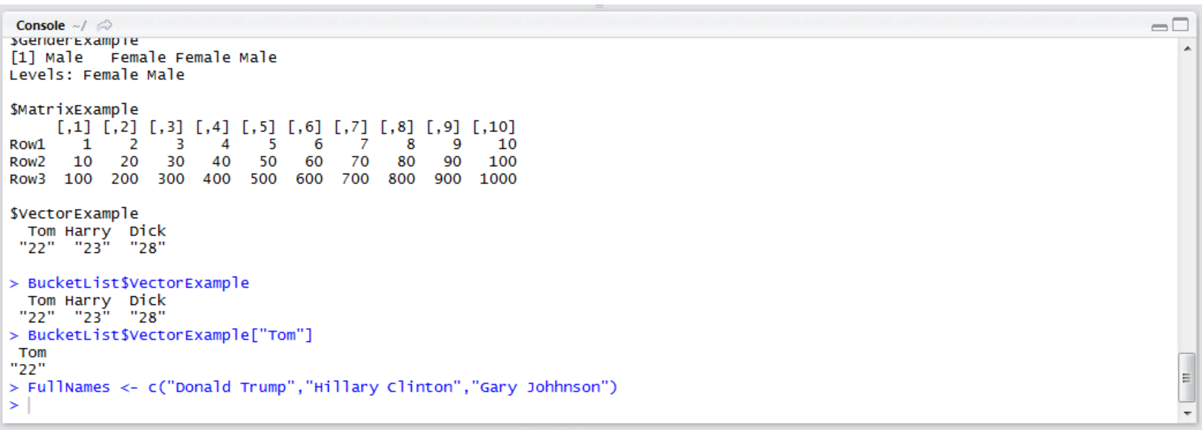
To create a data frame of customers, start by creating a vector of full names:

FullNames <- c("Donald Trump","Hilary Clinton"," Gary Johnson")



```
Matrix3Col <- cbind(Column1,Column2,Column3)
Matrix3Col
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ac
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
```

Run the line of script to console:



```
$GenderExample
[1] Male   Female Female Male
Levels: Female Male

$MatrixExample
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000

$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
>
```

Repeat for a Vector of FullAges:

FullAges <- c(70,69,50)

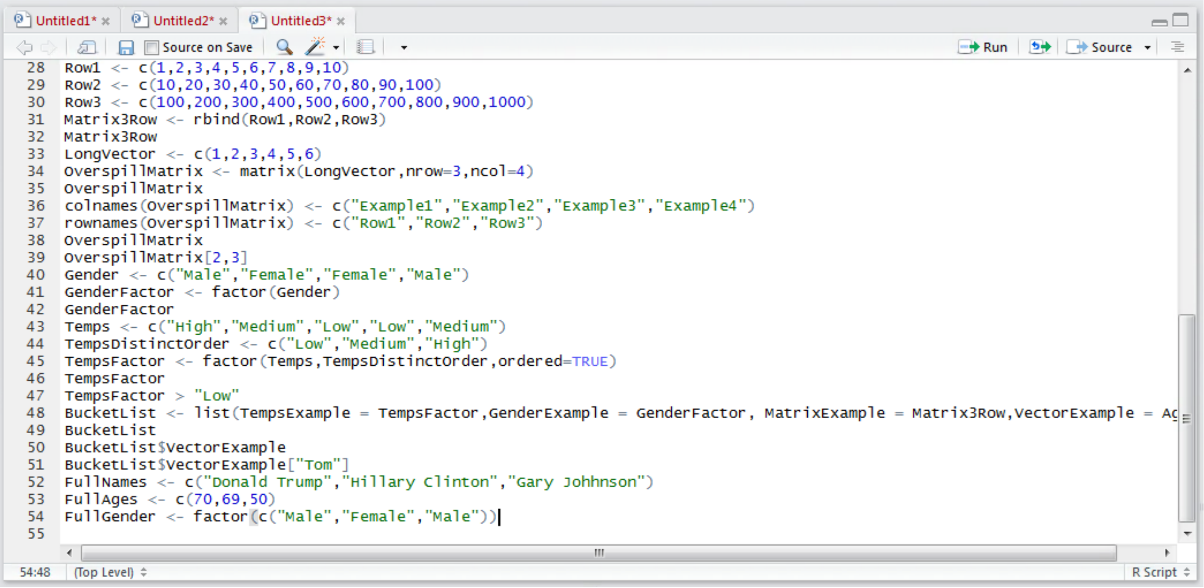
```
View(Matrix3Col)
Row1 <- c(1,2,3,4,5,6,7,8,9,10)
Row2 <- c(10,20,30,40,50,60,70,80,90,100)
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ag
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
FullAges <- c(70,69,50)
```

Run the line of script to console:

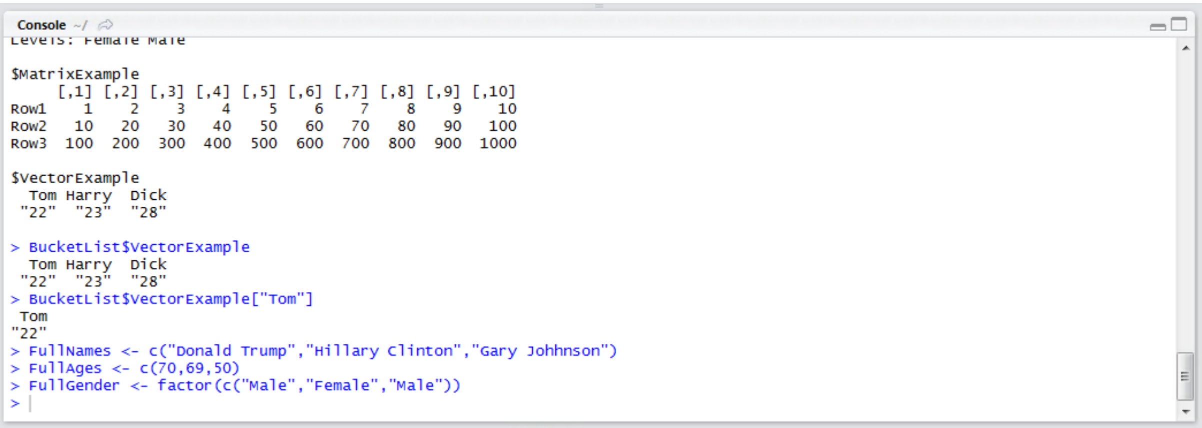
```
[1] Male   Female Female Male
Levels: Female Male

$MatrixExample
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000

$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
> FullAges <- c(70,69,50)
> 
```

Repeat for a Factor of FullGender, noting that the result of the c() function is being passed as the argument to thee factor() function:

FullGender <- factor(c("Male","Female","Male"))

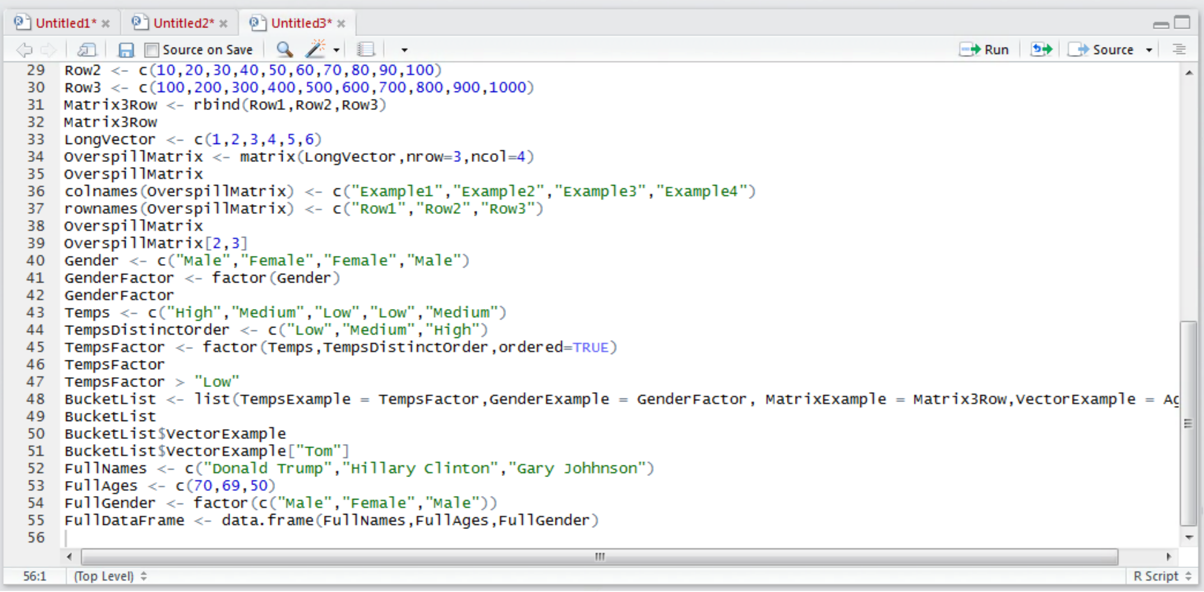


Run the line of script to console:

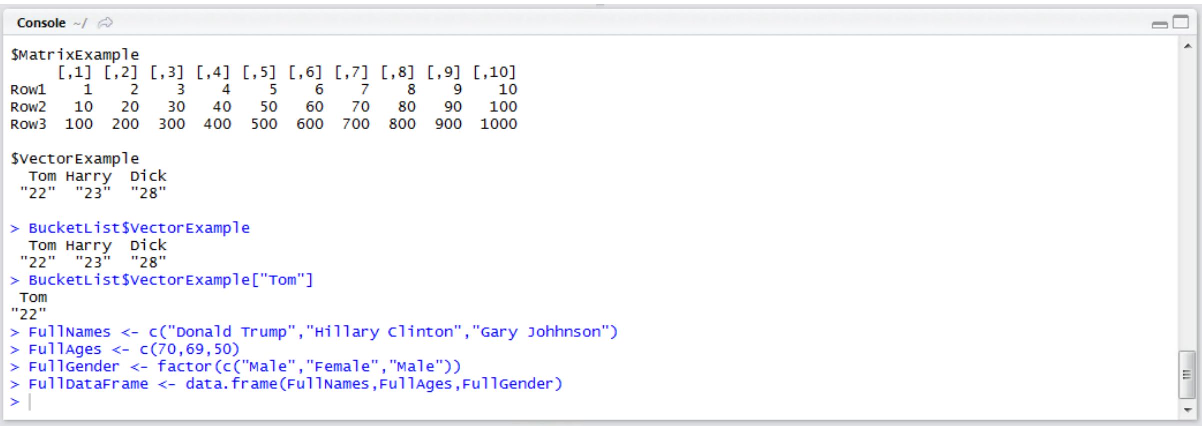


In a similar manner to both the c() function and the list() function, the data.frame() fuction takes Vectors or Factors of the same length and combines them into a Data Frame. As with the list() function it accepts a number of arguments in its advanced use, however, its most basic structure is the same as c(). To create a dataframe with default agurments type:

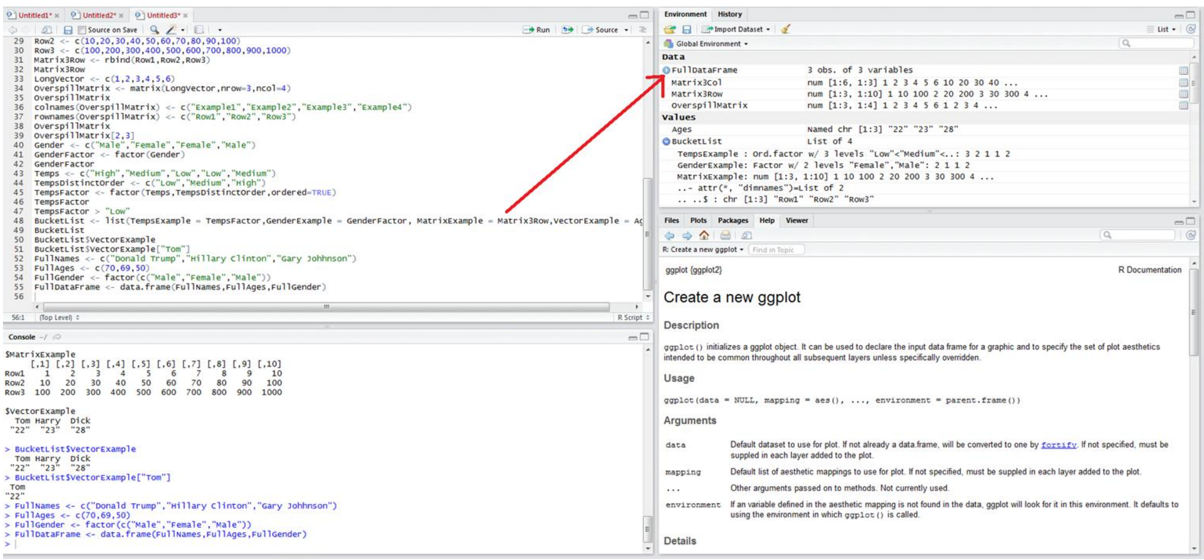
FullDataFrame <- data.frame(FullNames,FullAges,FullGender)



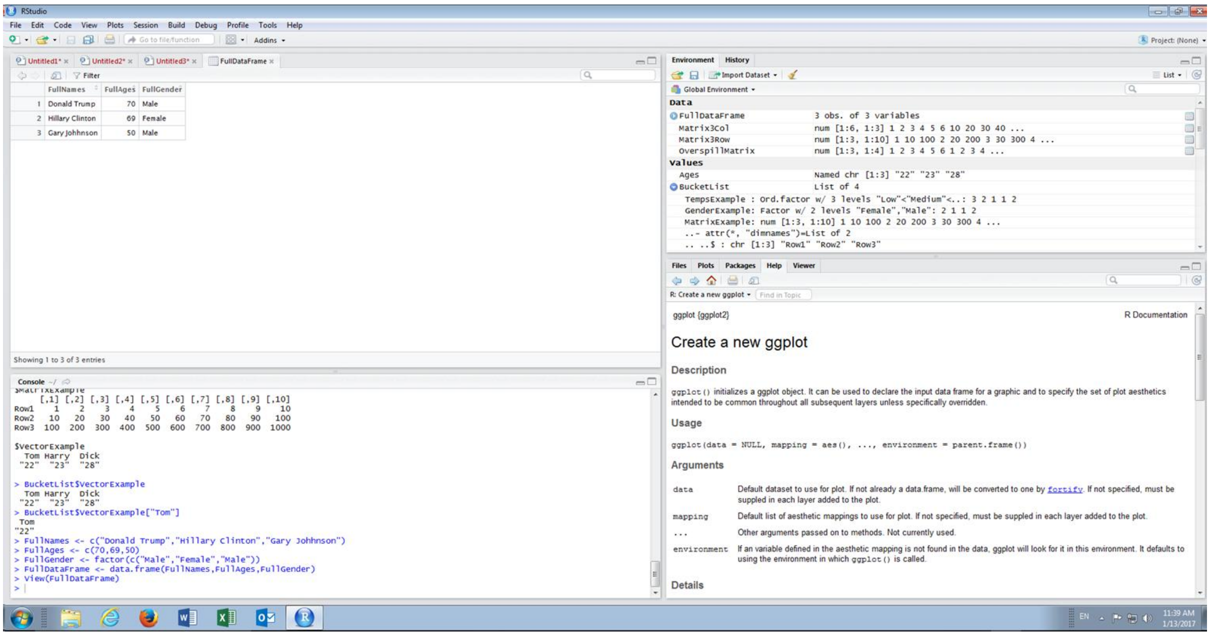
Run the line of script to console:



It can be observed that the data frame is now displayed in the environment pane under the data section and as such can be viewd in a similar manner to that set forth in procedure 27.



In this example a view is performed by a single click of the entry under the data section of the environment pane:



In a simiular manner to a Matrix, the Data Frame is expanded into the grid viewer section of RStudio as a table.

Procedure 19: Create a Data Frame from Names and stringsAsFactors.

As introduced previoulsy the data.frame() function, not unlike the list() function, has more flexibility to be able to create objects than the c() function.   As seems intuative it is possible to specify names explicity rather than take the names of the Vectors by default. There is an argument to the data.frame() function that can easy the burden of creating factors upon detection of chracter vectors in the for of the stringsAsFactors switch (although it is not always sensible to use it in the case of numeric prediction focus).

To create a Data Frame with specific names and disabling stringsAsFactors:

LabeledDataFrame <- data.frame(data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,stringsAsFactors = FALSE))

```
Untitled1* × Untitled3* ×
Source on Save                                   Run  Source
100,500,600,700,800,900,1000)
w1,Row2,Row3)

1,5,6)
rix(LongVector,nrow=3,ncol=4)

ix) <- c("Example1","Example2","Example3","Example4")
ix) <- c("Row1","Row2","Row3")


male","Female","Male")
(Gender)

ium","Low","Low","Medium")
:("Low","Medium","High")
Temps,TempsDistinctOrder,ordered=TRUE)


psExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ages)

le
le["Tom"]
 Trump","Hillary Clinton","Gary Johhnson")
)
("Male","Female","Male"))
frame(FullNames,FullAges,FullGender)
ta.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,stringsAsFactors = FALSE)

56:144  (Top Level) ÷                                                         R Script ÷
```

Run the line of script to console:

```
Console ~/
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Row1    1    2    3    4    5    6    7    8    9    10
Row2   10   20   30   40   50   60   70   80   90   100
Row3  100  200  300  400  500  600  700  800  900  1000

$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
> FullAges <- c(70,69,50)
> FullGender <- factor(c("Male","Female","Male"))
> FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
> LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,string
sAsFactors = FALSE)
> |
```

Return the Data Frame by typing:

```
Untitled1* × Untitled3* ×
Source on Save                                   Run  Source
Row3 <- c(100,200,300,400,500,600,700,800,900,1000)
Matrix3Row <- rbind(Row1,Row2,Row3)
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ag
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
FullAges <- c(70,69,50)
FullGender <- factor(c("Male","Female","Male"))
FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,str
LabeledDataFrame

57:17  (Top Level) ÷                                                          R Script ÷
```

Run the line of script to console:

```
$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
> FullAges <- c(70,69,50)
> FullGender <- factor(c("Male","Female","Male"))
> FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
> LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,string
sAsFactors = FALSE)
> LabeledDataFrame
  ExampleFullNames ExampleFullAges ExampleFullGender
1     Donald Trump              70              Male
2  Hillary Clinton              69            Female
3    Gary Johhnson              50              Male
> 
```

It can be observed that the column names have been correctly specified. Unless a factor has been specifically allocated it can be trusted that other chracter Vectors, such as FullName in this example, will not be transposed to factors automatically.

Procedure 20: Saving .Rdata to file.

Machine learning is predominentely a challenge of data abstraction – this is the shaping and moulding of data – and presenting it to advanced machine learning algorithms on a comodity basis. It follows that upon having spent time and effort creating an elaborate Data Frame, it likely that it will need to be saved for future use (if only to avoid the computational expense of recreating it).

The save() function exists for the purpose of saving most objects that can be created and populated with data to a file in the working directory. It is a very important part to deploying models on a real-time basis.

To save the Data Frame LabeledDataFrame and BucketList to a specified file by the name of "Example.RData":

save(LabeledDataFrame,BucketList,file = "Example.RData")

```
Matrix3Row
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ag
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
FullAges <- c(70,69,50)
FullGender <- factor(c("Male","Female","Male"))
FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,str
LabeledDataFrame
save(LabeledDataFrame,BucketList,file="Example.RData")

```

Run the line of script to console:

```
Console ~/
  Tom Harry  Dick
 "22"  "23"  "28"

> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
> FullAges <- c(70,69,50)
> FullGender <- factor(c("Male","Female","Male"))
> FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
> LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,string
sAsFactors = FALSE)
> LabeledDataFrame
  ExampleFullNames ExampleFullAges ExampleFullGender
1     Donald Trump              70              Male
2  Hillary Clinton              69            Female
3     Gary Johhnson             50              Male
> save(LabeledDataFrame,BucketList,file="Example.RData")
> 
```

A file titled Example.RData is not written out to the Working Directory. To remind the working directory:

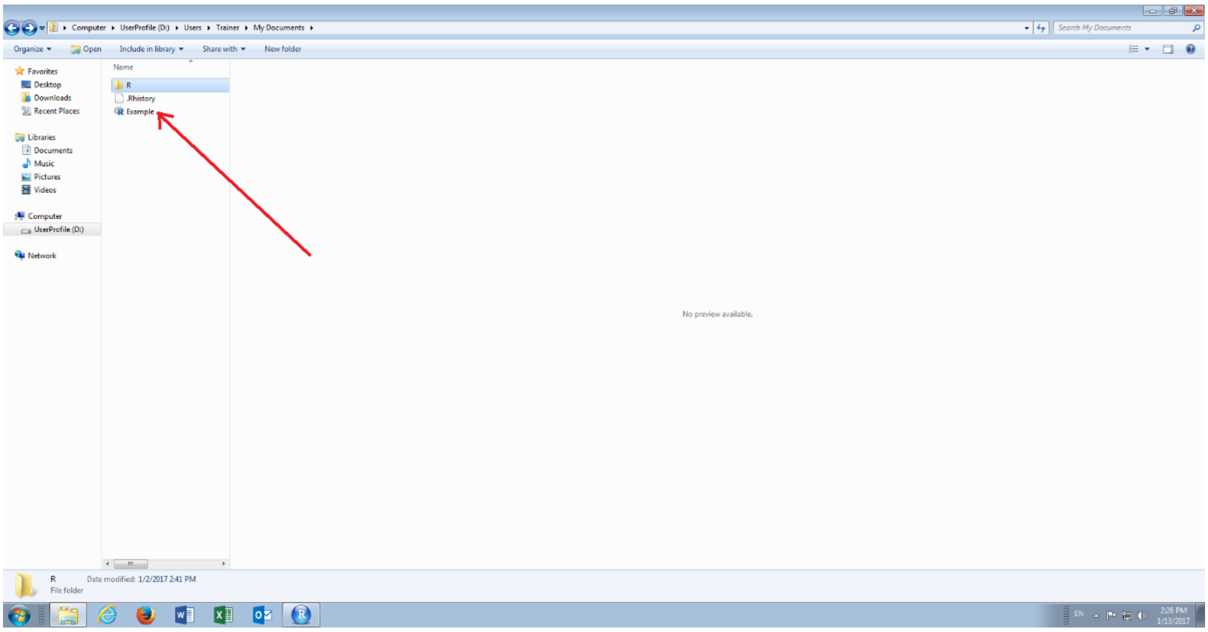
getwd()

```
Untitled1* × Untitled3* ×
Source on Save                                   Run    Source
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ag
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
FullAges <- c(70,69,50)
FullGender <- factor(c("Male","Female","Male"))
FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,str
LabeledDataFrame
save(LabeledDataFrame,BucketList,file="Example.RData")
getwd()

60:1   (Top Level)                                                R Script
```

Run the line of script to console:

```
Console ~/
> BucketList$VectorExample
  Tom Harry  Dick
 "22"  "23"  "28"
> BucketList$VectorExample["Tom"]
 Tom
"22"
> FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
> FullAges <- c(70,69,50)
> FullGender <- factor(c("Male","Female","Male"))
> FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
> LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,string
sAsFactors = FALSE)
> LabeledDataFrame
  ExampleFullNames ExampleFullAges ExampleFullGender
1     Donald Trump              70              Male
2  Hillary Clinton              69            Female
3     Gary Johhnson             50              Male
> save(LabeledDataFrame,BucketList,file="Example.RData")
> getwd()
[1] "D:/Users/Trainer/Documents"
> 
```

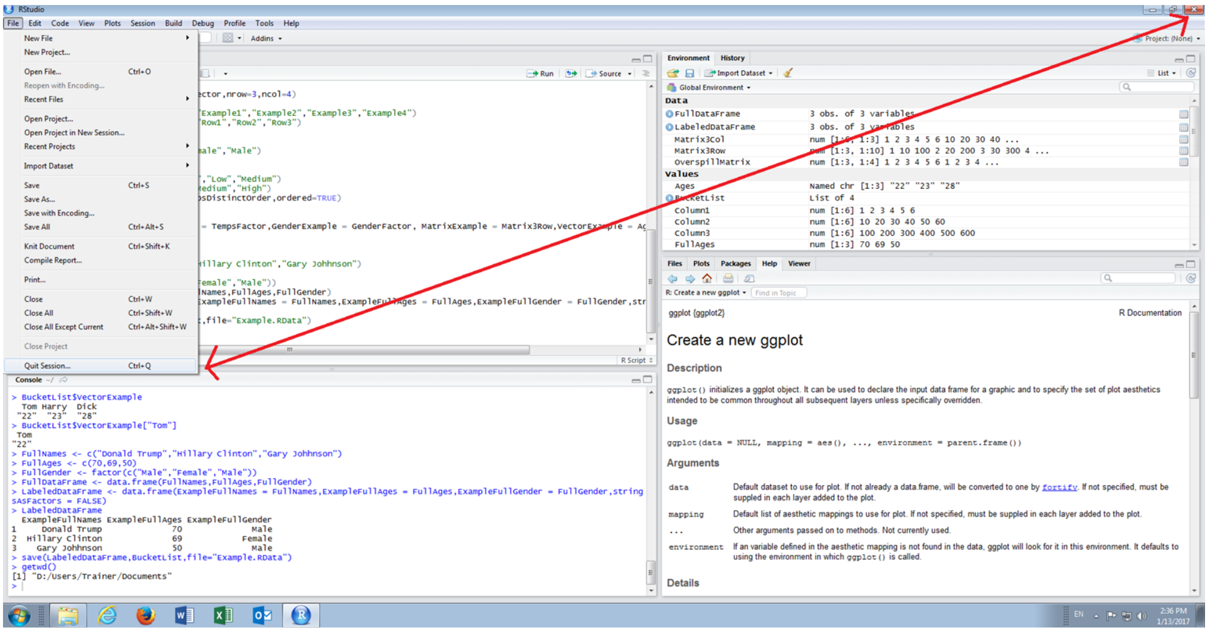
Having identified the working directory, navigate to the same in windows explorer:



The saved file is clearly visible in this directory ready for real-time deployment or being reloaded to an R session.

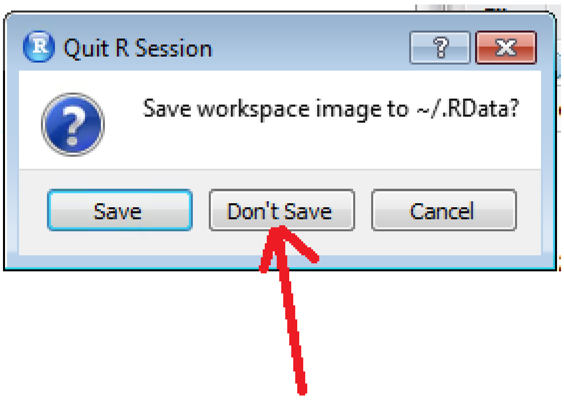
Procedure 21: Loading .Rdata from file.

To fully demonstrate the process of loading objects from an RData file fully close down RStudio by clicking File, then upon the menu expanding, clicking Quit Session or by clicking on the close button in the top right hand corner:

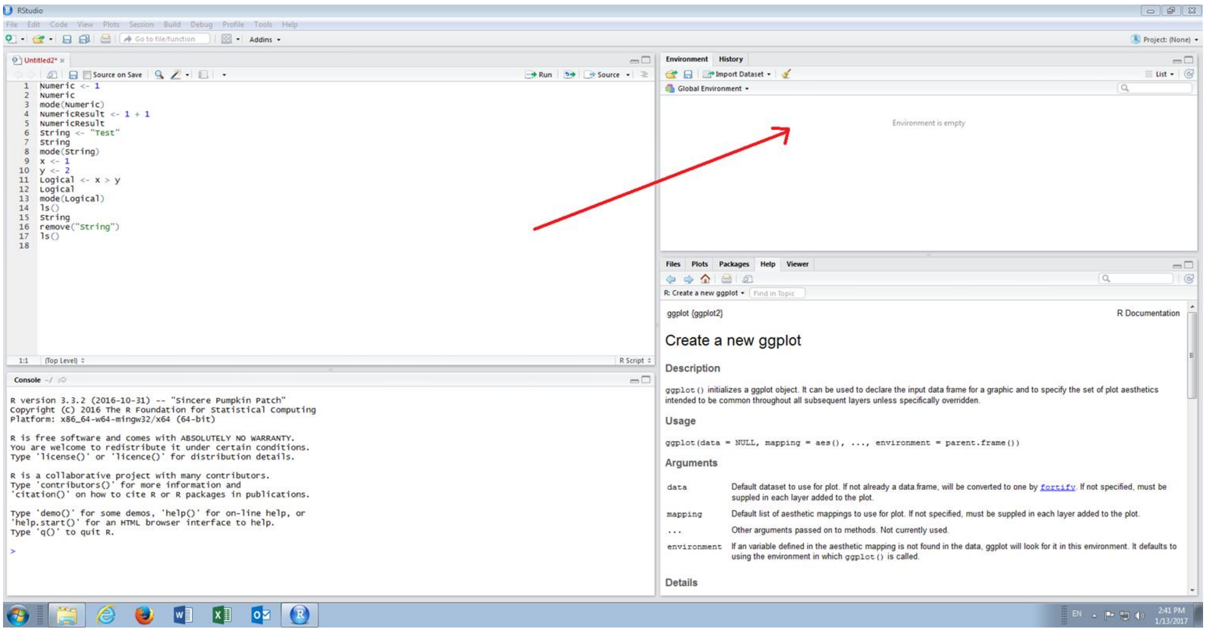


As expected, similarly to procedure 38, confirmation will be sought about the treatment of the current session. Elect not to save the session by clicking "Don't Save":

JUBE



Upon termination of RStudio, simply reload as specified in procedure 5:



It can be seen that there are no objects loaded. Assuming the working directory is unchanged, to load the objects saved in procedure 38, simply type:

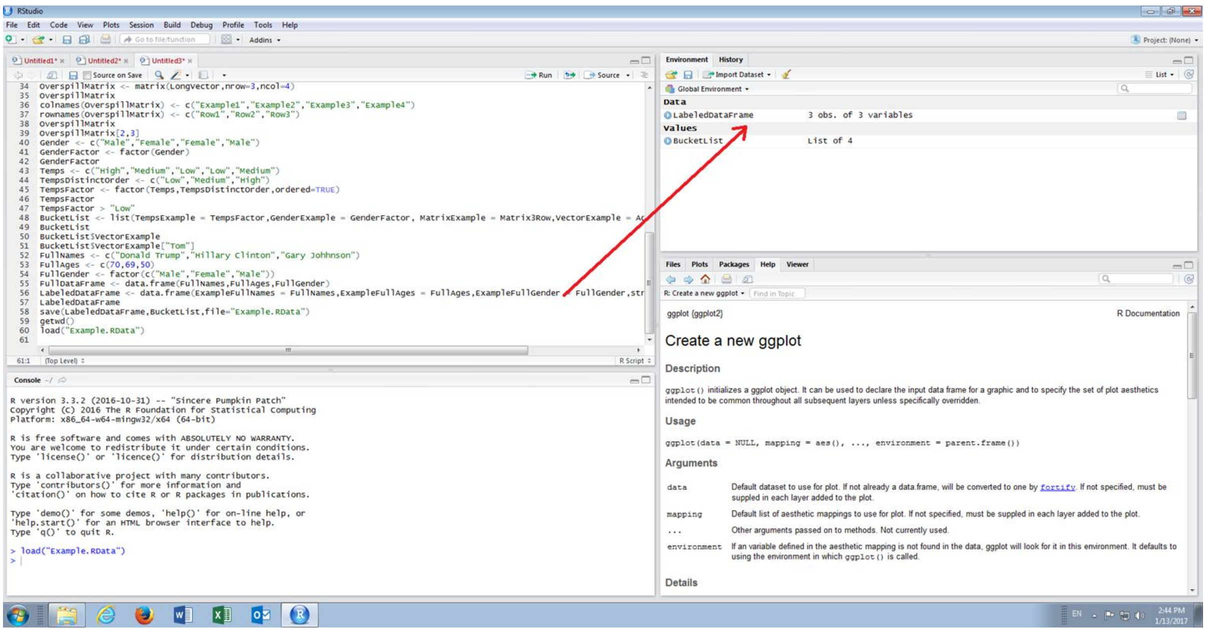
load("Example.RData")

```
LongVector <- c(1,2,3,4,5,6)
OverspillMatrix <- matrix(LongVector,nrow=3,ncol=4)
OverspillMatrix
colnames(OverspillMatrix) <- c("Example1","Example2","Example3","Example4")
rownames(OverspillMatrix) <- c("Row1","Row2","Row3")
OverspillMatrix
OverspillMatrix[2,3]
Gender <- c("Male","Female","Female","Male")
GenderFactor <- factor(Gender)
GenderFactor
Temps <- c("High","Medium","Low","Low","Medium")
TempsDistinctOrder <- c("Low","Medium","High")
TempsFactor <- factor(Temps,TempsDistinctOrder,ordered=TRUE)
TempsFactor
TempsFactor > "Low"
BucketList <- list(TempsExample = TempsFactor,GenderExample = GenderFactor, MatrixExample = Matrix3Row,VectorExample = Ac
BucketList
BucketList$VectorExample
BucketList$VectorExample["Tom"]
FullNames <- c("Donald Trump","Hillary Clinton","Gary Johhnson")
FullAges <- c(70,69,50)
FullGender <- factor(c("Male","Female","Male"))
FullDataFrame <- data.frame(FullNames,FullAges,FullGender)
LabeledDataFrame <- data.frame(ExampleFullNames = FullNames,ExampleFullAges = FullAges,ExampleFullGender = FullGender,str
LabeledDataFrame
save(LabeledDataFrame,BucketList,file="Example.RData")
getwd()
load("Example.RData")
```

Run the line of script to console:

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> load("Example.RData")
> 
```

The objects saved previously are promptly loaded and available in the environment pane of RStudio and by implication available for recall in scirpts and \ or the console.



As R has several programatic implementaions, such as R.net which is used for real-time invocation, the saving and loading of R seesions provides a useful means to be able to deploy objects.

Module 4: Loading, Shaping and Merging Data.

Abstraction, the process of shaping and moulding raw data to enhance relevance prior to it being presented to machine learning algorithms, is the cornerstone of the methodologies put forward in these procedures.

The procedures that follow set out the means to load data into R, and when this data resides in R, sets forth procedures to shape and mould the data in as part of abstraction.

Most generally in Jube procedures and methodology Abstraction is offloaded to Relational Database Management platforms, the shaping and moulding of data in R tends to be to augment these core datasets.

Procedure 1: Using Numeric Functions to create a Horizontal Abstraction.

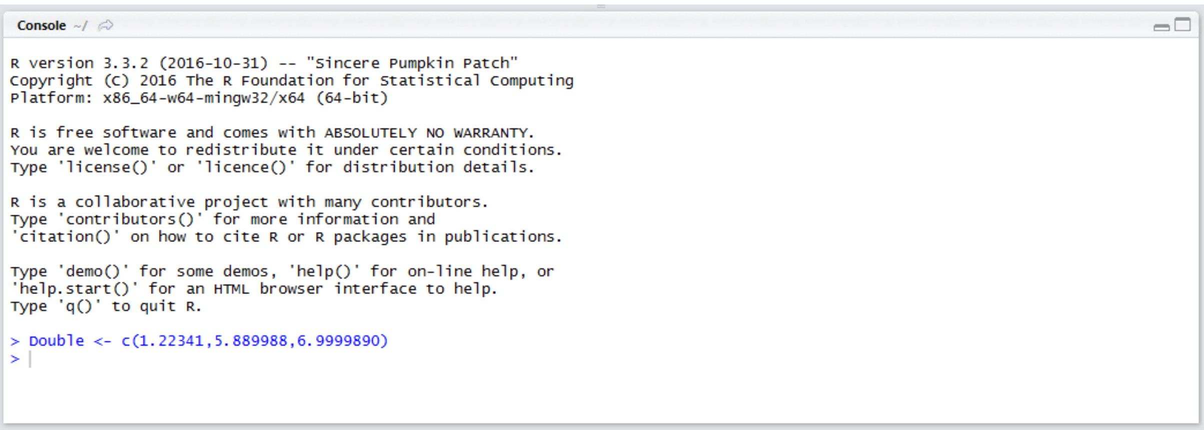
As introduced R has a plethora of procedures that facilitate the creation of Vectors and Matrices, furthermore there are base numeric operators which facilitate:

- + Addition.
- - Subtraction.
- * Multiplication.
- / Division.
- %% Exponent.
- ^ Power Of.

Functions also provide the ability to perform mathematical operations. In this example, a vector of double values will be created then rounded. Create a new script and start by creating a vector containing double values:

Double <- c(1.22341,5.889988,6.9999890)

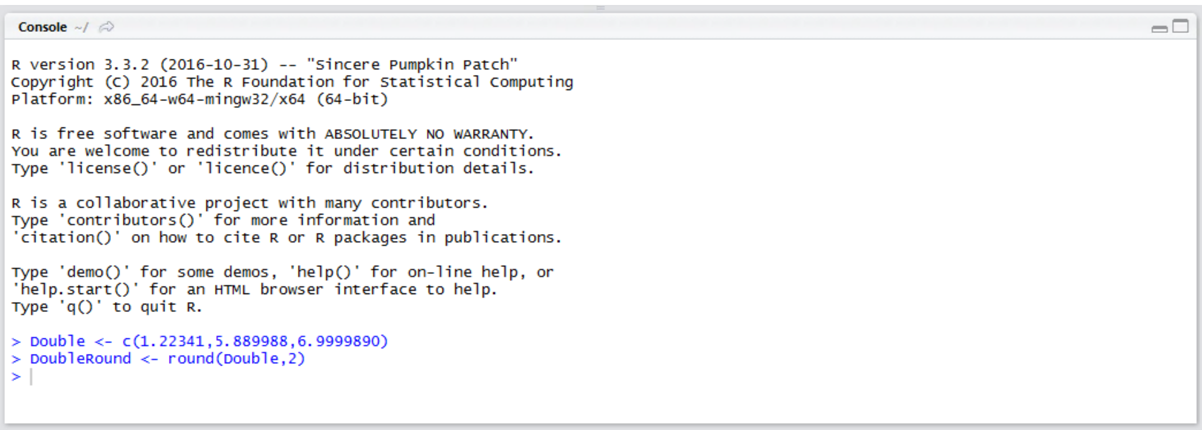
Run the line of script to console:



Use the round() function, which takes two arguments of value and digits, to round the Double vector to two decimal places assigning that vector:

DoubleRound <- round(Double,2)

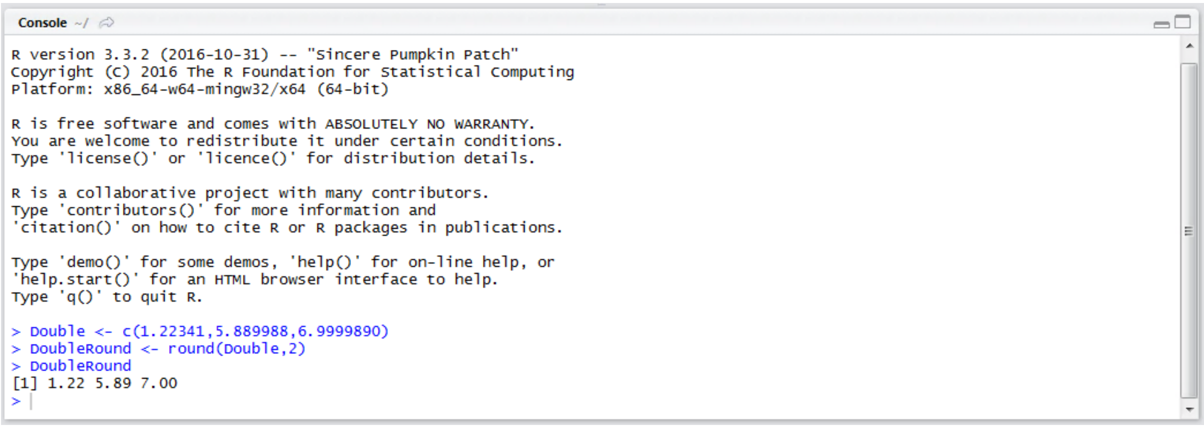
Run the line of script to console:



Write out the DoubleRound vector by typing:

DoubleRound

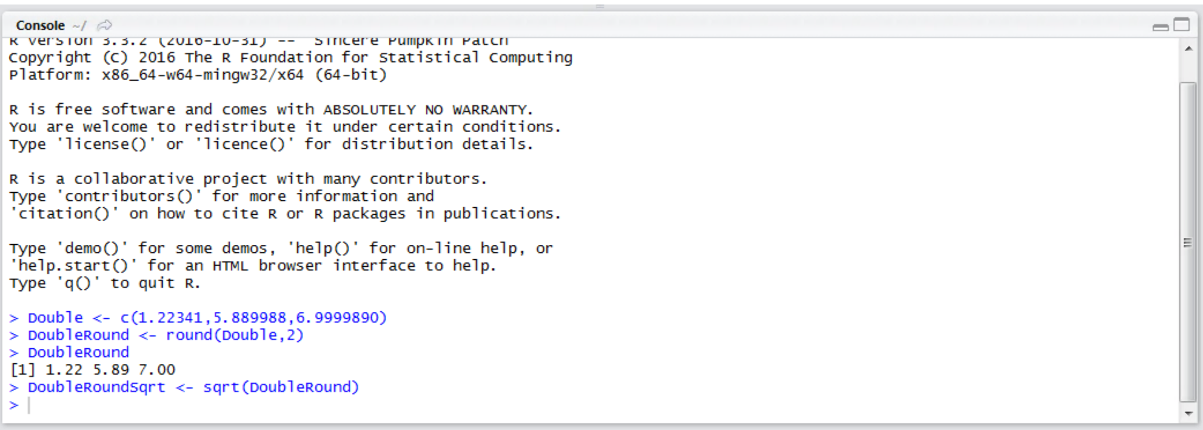
Run the line of script to console:



It can be observed that the vector has been rounded to two decimal places. By way of further abstraction find the square root:

DoubleRoundSqrt(VectorRound)

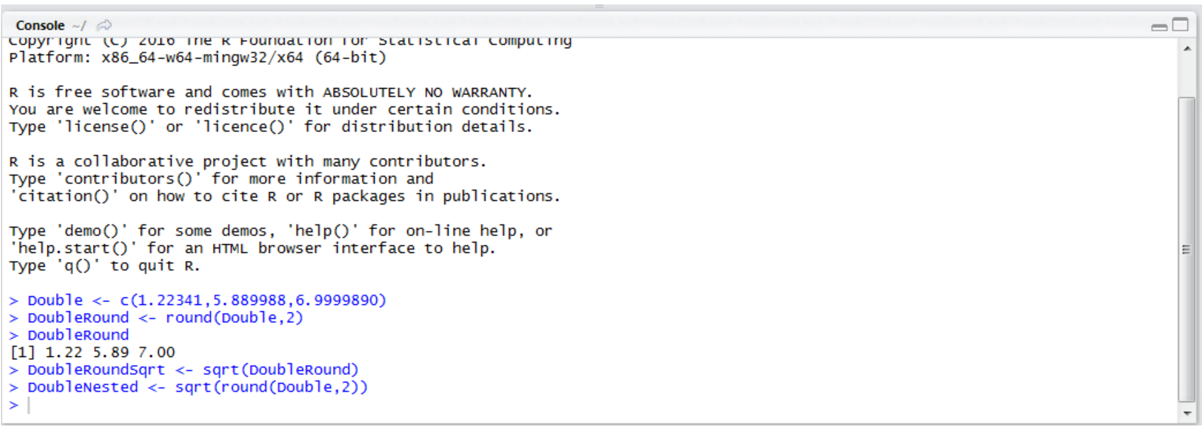
Run the line of script to console:



A more concise way to create a line of script relying on several functions, could include nesting the functions:

DoubleNested <- sqrt(round(Double,2))

Run the line of script to console:



It can be observed that with the help of several R numeric functions that complex horizontal abstractions can take place.

Procedure 2: Extracting a substring from a string, testing logically and presenting for machine learning.

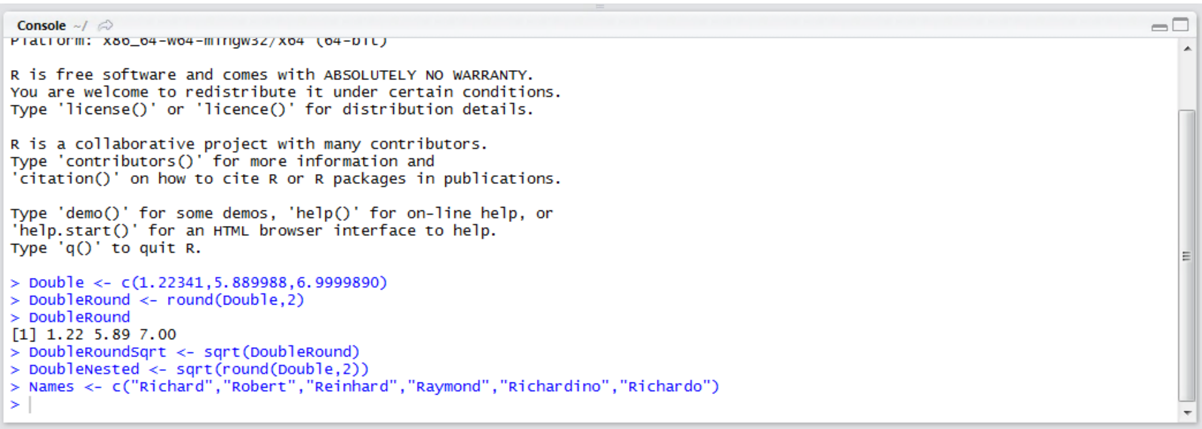
In Horizontal Abstraction, it is quite common to have the requirement to inspect a string of data looking for an occurrence (or pattern) and return a logical value that can be used in machine learning.

In this example, a string will be inspected and return a 1 in the event that the string "Richard" is present.

Firstly, create a vector of name strings by typing:

Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")

Run the line of script to console:

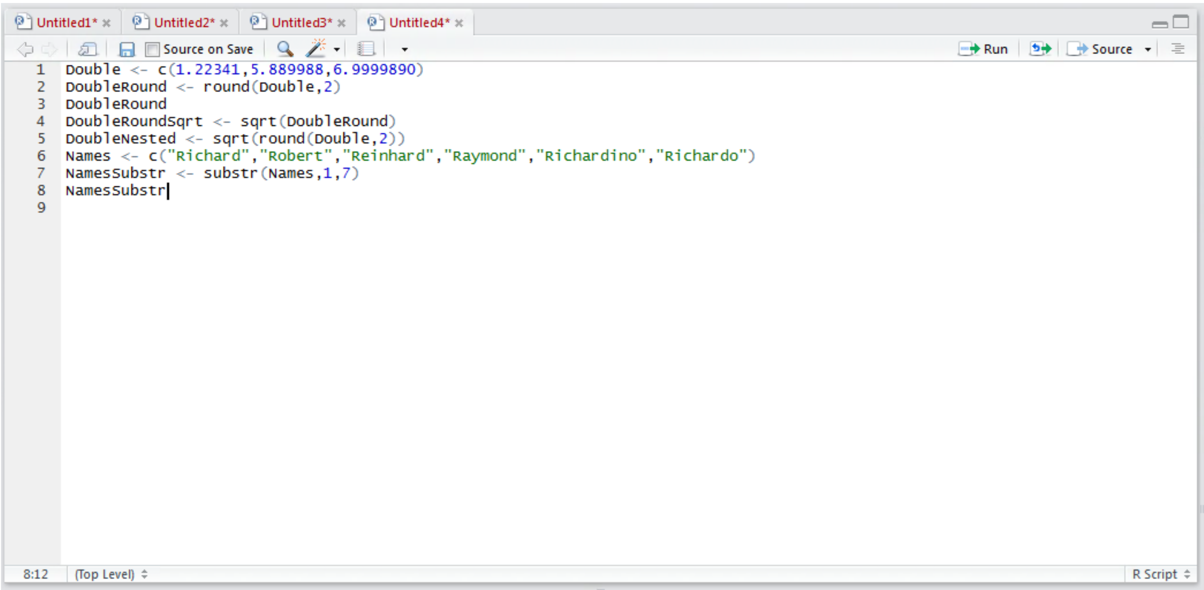


Use the substr() function to create a vector of the first 7 characters of the value contained in the Names vector, by typing:

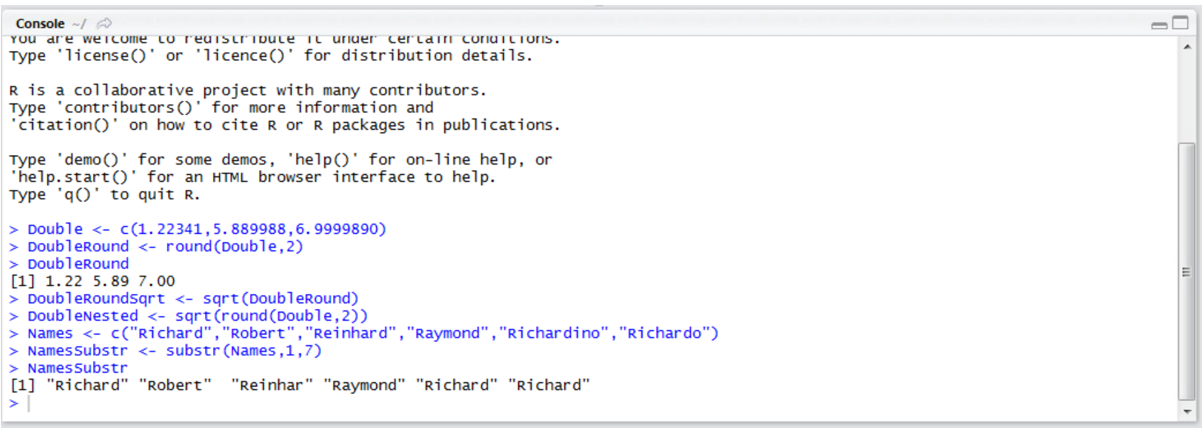
NamesSubstr <= substr(Names,1,7)

Write the NamesSubstr vector:

NamesSubstr

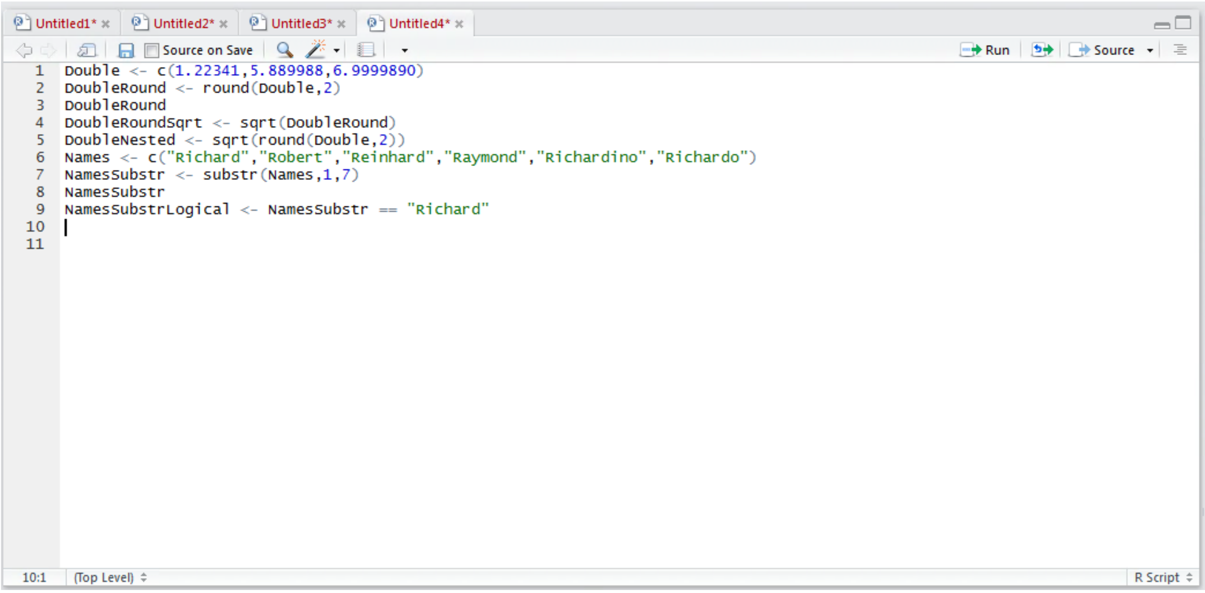


Run the line of script to console:



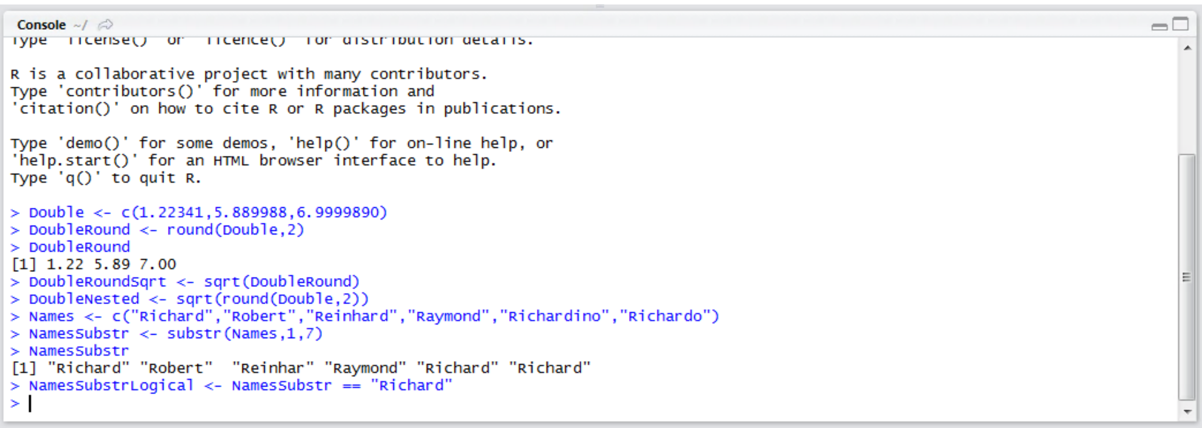
The question being posed is whether the first characters of the name is equal to "Richard". To perform this evaluation, create a logical vector from the NamesSubstr vector by typing:

NamesSubstrLogical <- NamesSubstr == "Richard"



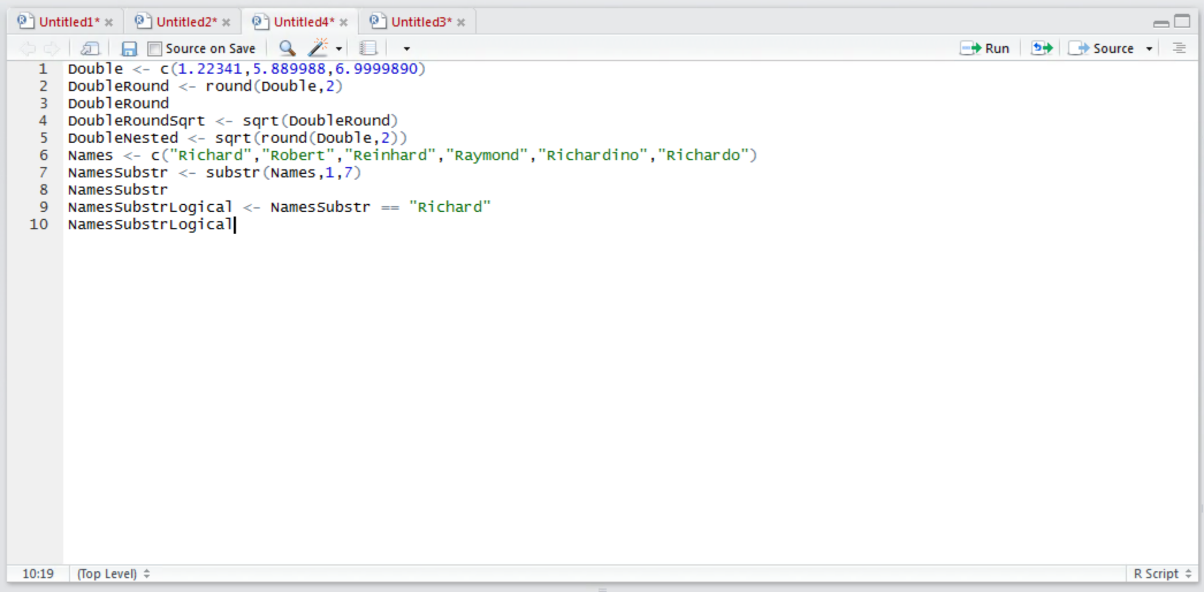
Notice how a double equals sign is used to eliminate confusion between evaluation and assignment.

Run the line of script to console:

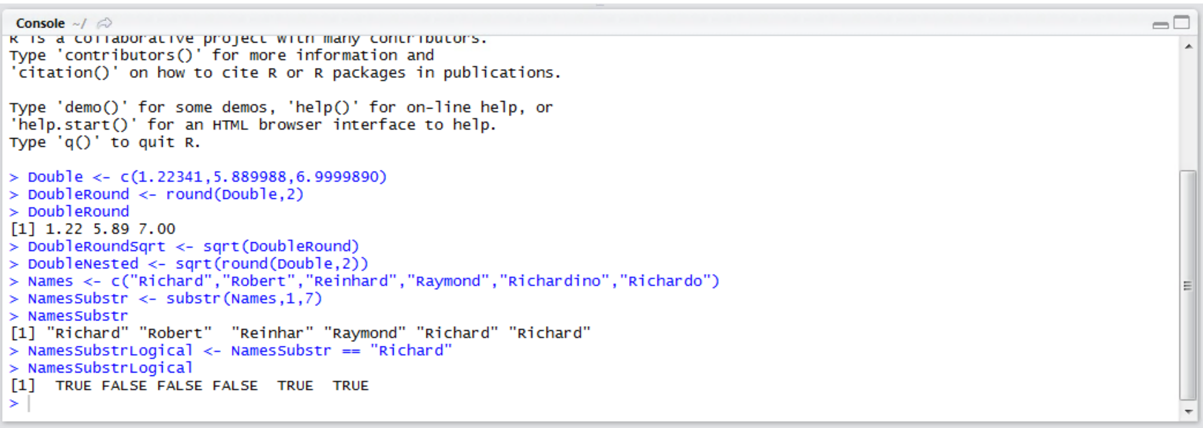


Write the logical vector out to console by typing:

NamesSubstrLogical



Run the line of script to console:



The character notion of TRUE or FALSE cannot be used in machine learning readily (you can't multiply by text) and it follows that these values should be converted to a numeric value using the as.numeric() function, typing:

NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
```

Run the line of script to console:

```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Double <- c(1.22341,5.889988,6.9999890)
> DoubleRound <- round(Double,2)
> DoubleRound
[1] 1.22 5.89 7.00
> DoubleRoundSqrt <- sqrt(DoubleRound)
> DoubleNested <- sqrt(round(Double,2))
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
>
```

Write the newly created vector to console by typing:

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumeric
```

Run the line of script to console:

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumeric
```

A more concise line of script nesting the functions might be:

NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
```

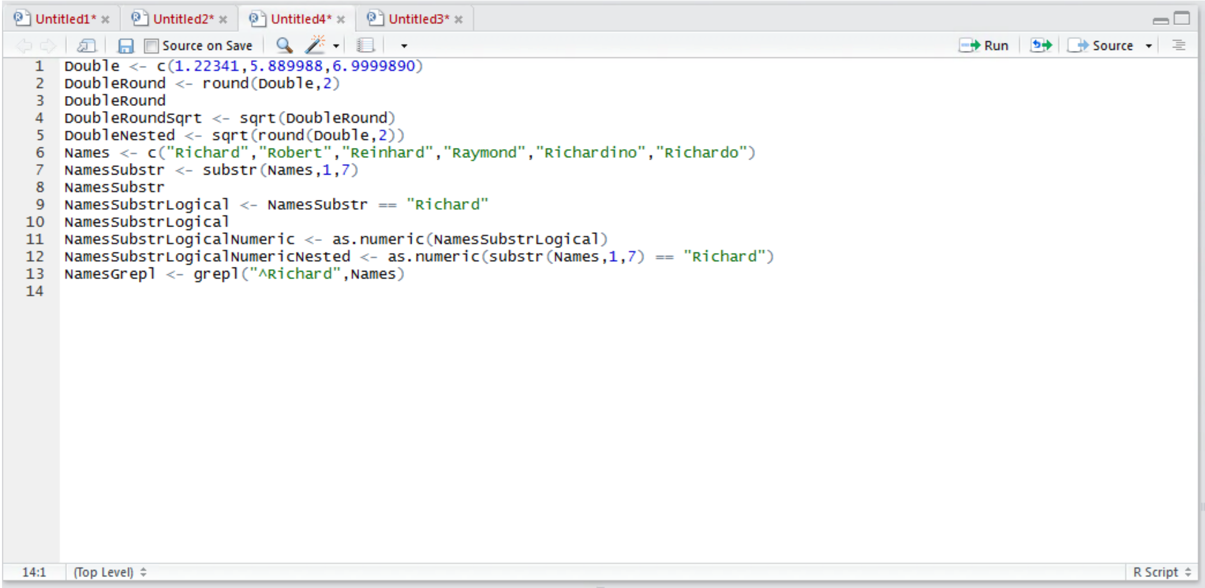
An alternative approach might be converting the logical vector to a factor as explained in procedure 32:

Procedure 3: Searching with Regular Expressions.

In procedure 41 the substr() function was used to search for any occurrence of the string "Richard". The substr() is a very limited function and assumes a certain amount of structure exists in the base string. The grepl() function allows for the searching of a character string with regular expressions rather than specific location based arguments. Regular Expressions are a sequence of symbols and characters expressing a string, or pattern, describing a search within a longer piece of text. Regular Expressions can be quite complex but they are extraordinarily powerful for string matching.

This procedure sets out to replicate the substr() function using Regular Expressions and the grepl() function, searching for any string that starts with "Richard" using the ^ symbol:

NamesGrepl <- grepl(^Richard,NamesSubstr)



```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
```

Run the line of script to console:

```
help.start() for an HTML browser interface to help.
Type 'q()' to quit R.

> Double <- c(1.22341,5.889988,6.9999890)
> DoubleRound <- round(Double,2)
> DoubleRound
[1] 1.22 5.89 7.00
> DoubleRoundSqrt <- sqrt(DoubleRound)
> DoubleNested <- sqrt(round(Double,2))
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumeric
[1] 1 0 0 0 1 1
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
>
```

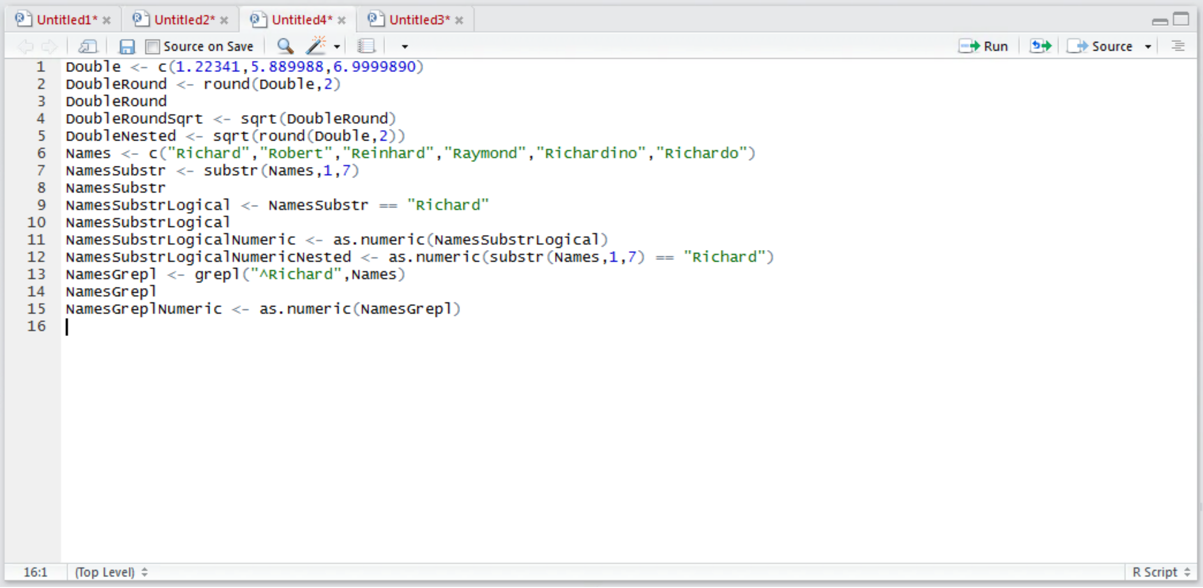
Write the NamesGrepl vector out to console by typing:

NamesGrepl

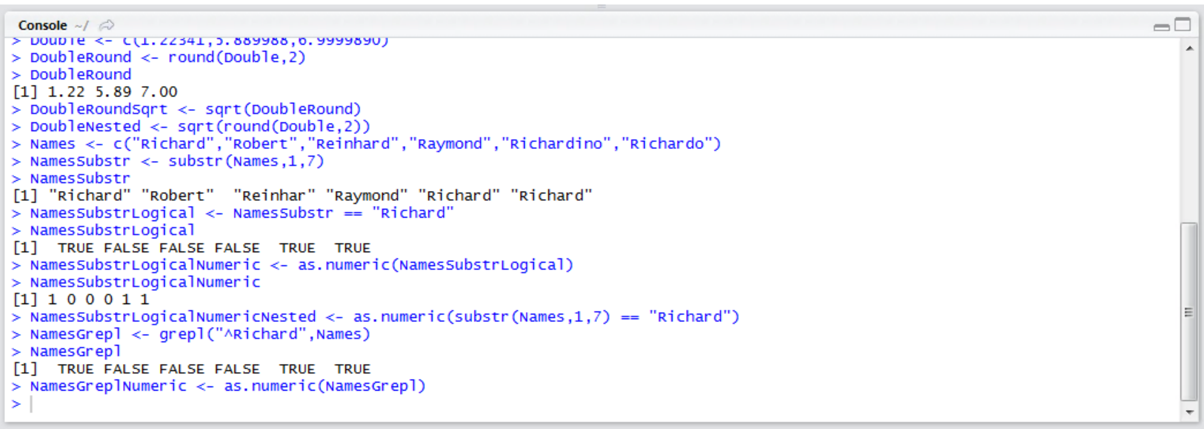
```
> Double <- c(1.22341,5.889988,6.9999890)
> DoubleRound <- round(Double,2)
> DoubleRound
[1] 1.22 5.89 7.00
> DoubleRoundSqrt <- sqrt(DoubleRound)
> DoubleNested <- sqrt(round(Double,2))
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumeric
[1] 1 0 0 0 1 1
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
>
```

It can be observed that any name string starting with "Richard" has been returned as a logical vector. To make this abstraction useful for machine learning it is a simple matter of transforming it to a numeric vector by typing:

NameGreplNumeric <- as.numeric(NamesGrepl)



Run the line of script to console:



Write out the NamesGrepNumeric vector by typing:

NamesGrepNumeric

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
```

Run the line of script to console:

```
> DoubleRound
[1] 1.22 5.89 7.00
> DoubleRoundSqrt <- sqrt(DoubleRound)
> DoubleNested <- sqrt(round(Double,2))
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumeric
[1] 1 0 0 0 1 1
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
>
```

It can be seen that this vector is now more appropriate for machine learning. Nesting the functions, the procedure could be created more sucinctly by typing:

NamesGrepNumericNested <- as.numeric(grepl("^Richard",Names))

```r
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
```

Procedure 4: Create a Date with a specific Date and Time format.

Dates have rather special treatment in R, not least that data can be presented in raw data in a variety of formats (e.g. DDMMYYYY, DD/MM/YYYY). The date data type in R exists for the purpose of interacting and manipulating dates.

A vector of dates would start out as a character vector:

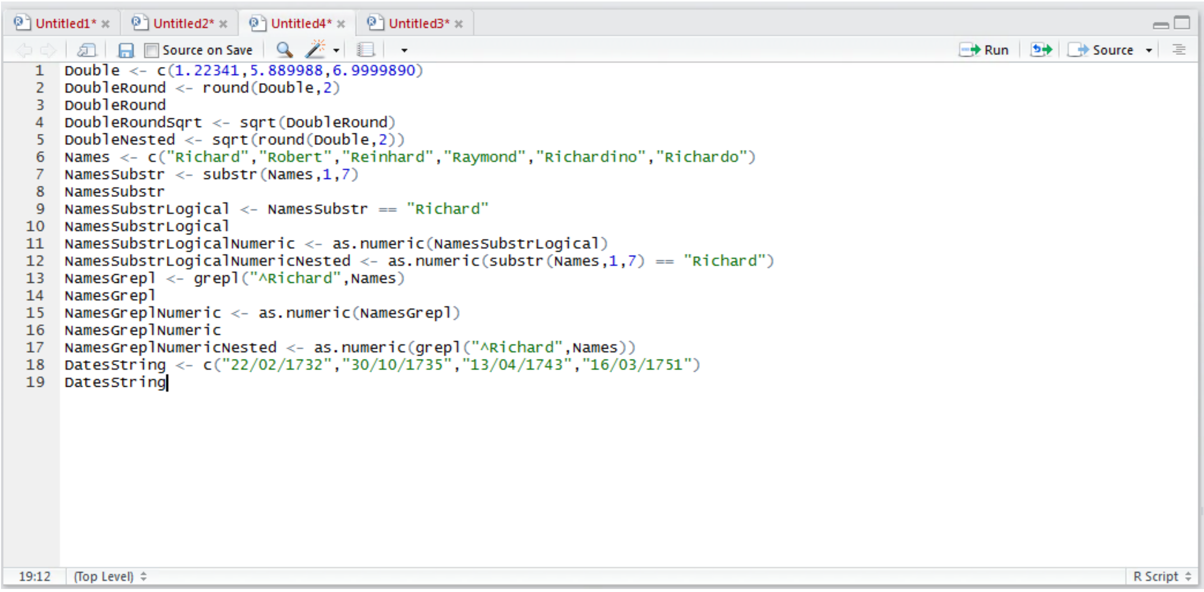
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")

```r
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
```

Run the line of script to console:

```
> DoubleRoundSqrt <- sqrt(DoubleRound)
> DoubleNested <- sqrt(round(Double,2))
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumeric
[1] 1 0 0 0 1 1
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> 
```

It can be observed that the dates are of the form charterer by typing:



```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
DatesString
```

Run the line of script to console:

```
> Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumeric
[1] 1 0 0 0 1 1
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> 
```

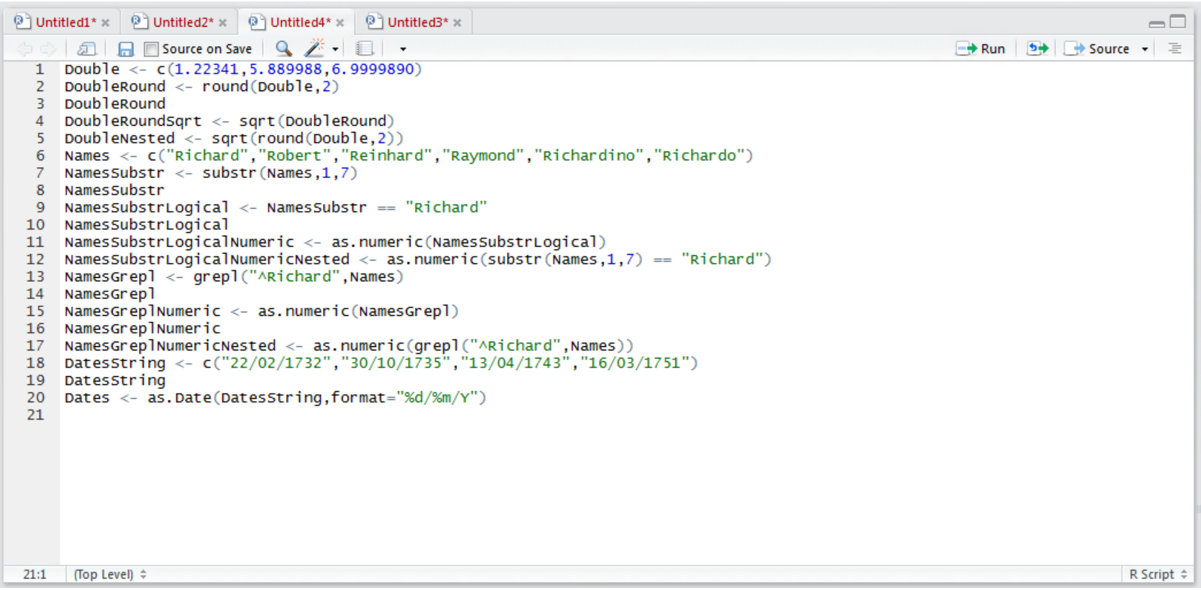
To convert the DatesString vector to the correct data type, R needs to know where to find the year component, the day component and the month component while knowing how to separate the elements. The following tokens specify the components:

- %Y is a four digit number.
- %y is a two digit number.

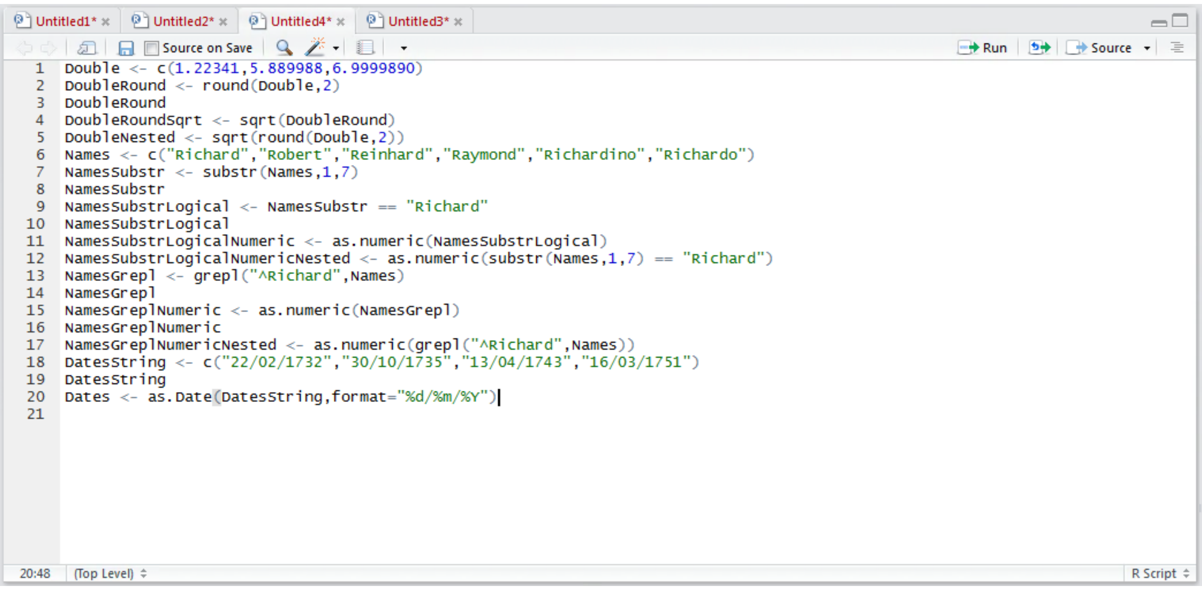
- %m is the month as a number.
- %d is the day as a number.
- %b is a short month (such as Jan).
- %B is a long month (such as January)

Outside of the % tokenisation characters can be specified that should be excluded in the overall tokenisation. To convert the character string vector of dates to a date vector type:

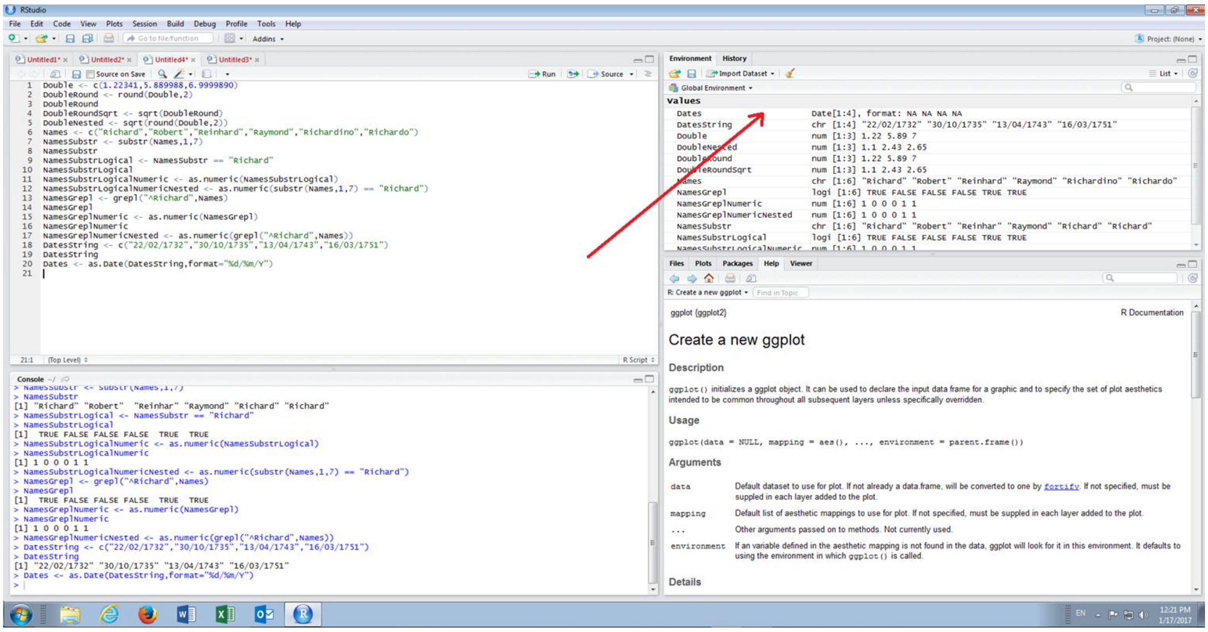
Dates <- as.Date(DatesString,format="%d/%m/%Y")



Run the line of script to console:

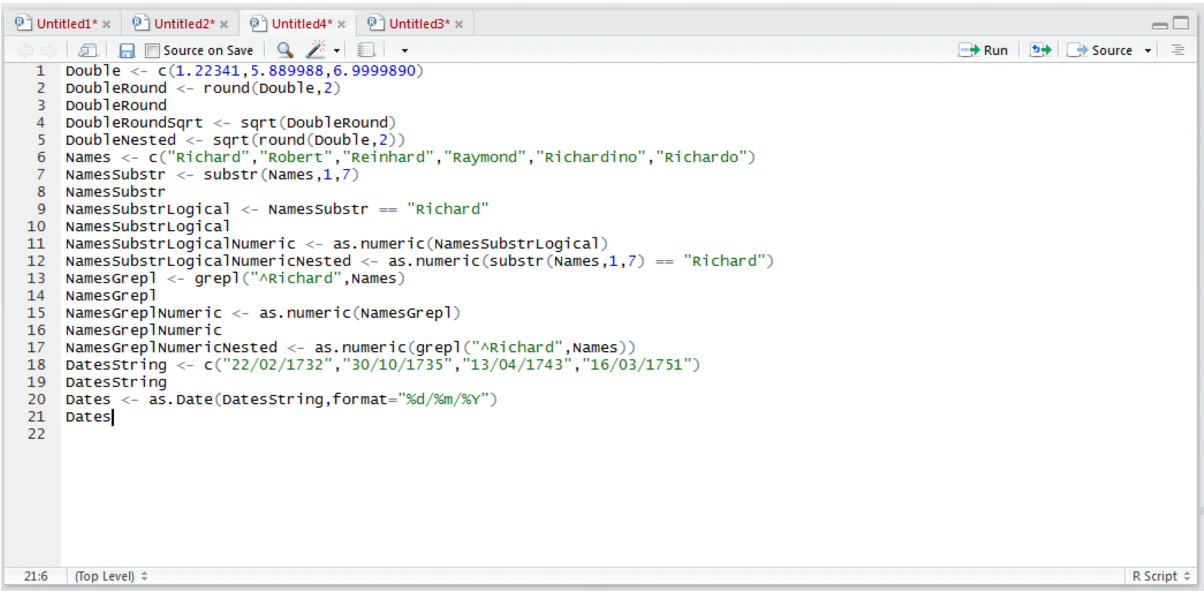


It can be observed that the Dates vector has been created in the environment pane:



Naturally the dates vector can be written out to the console by typing:

Dates



Run the line of script to console:

```
> NamesSubstr <- substr(Names,1,7)
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> Dates <- as.Date(DatesString,format="%d/%m/%Y")
> Dates
[1] "1732-02-22" "1735-10-30" "1743-04-13" "1751-03-16"
> 
```

Procedure 5: Perform Date Arithmetic.

Upon a date object, having been created it is possible to perform arithmetic on the dates. In this example one day is going to be added to the dates in the vector. To add a day to each value in vector type:

DatesPlusOne <- Dates + 1

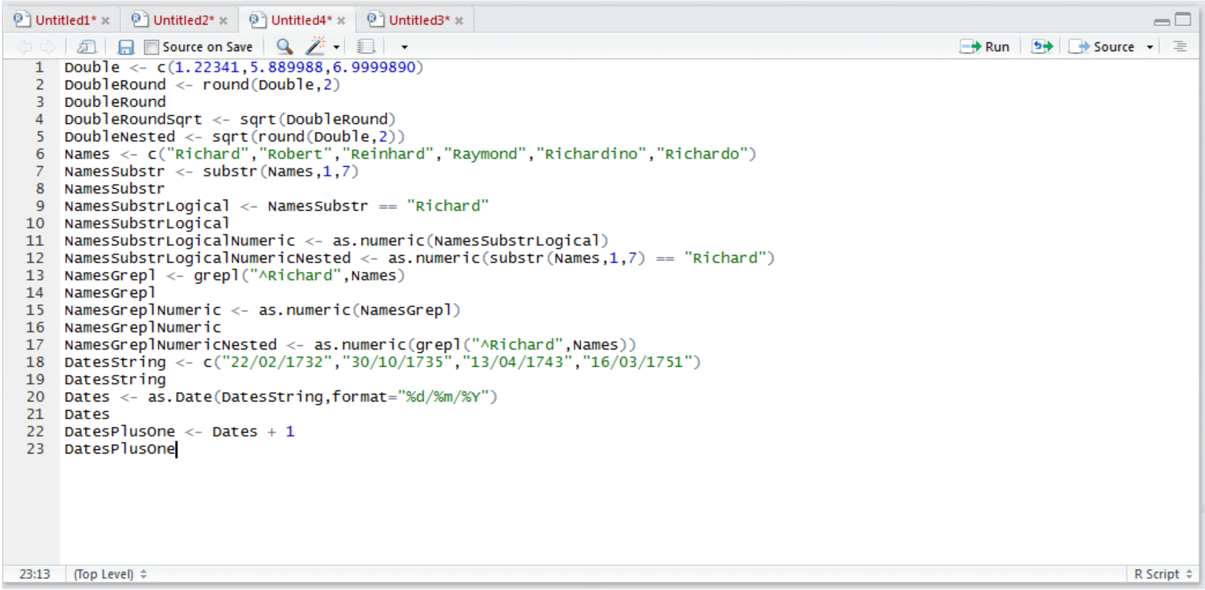
```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
DatesString
Dates <- as.Date(DatesString,format="%d/%m/%Y")
Dates
DatesPlusOne <- Dates + 1
```

Run the line of script to console:

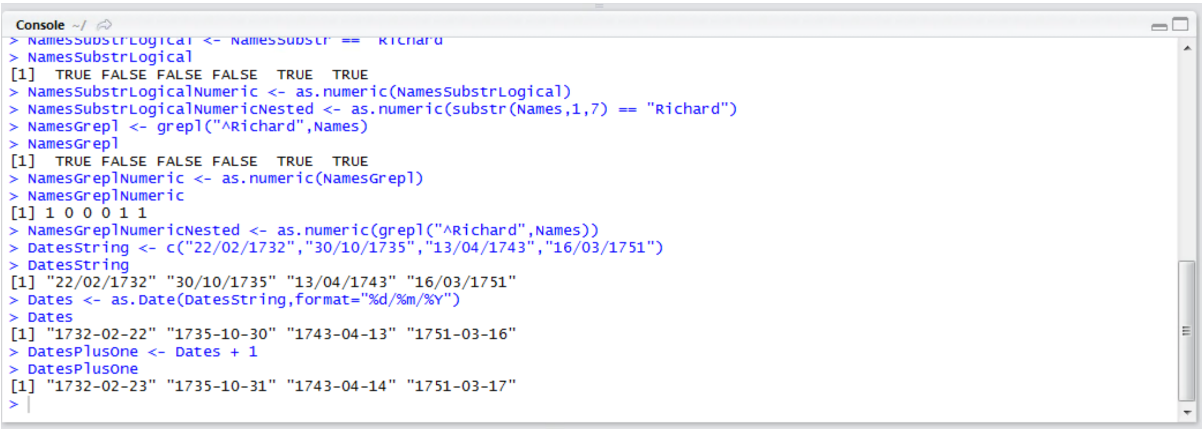
```
> NamesSubstr
[1] "Richard" "Robert"  "Reinhar" "Raymond" "Richard" "Richard"
> NamesSubstrLogical <- NamesSubstr == "Richard"
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> Dates <- as.Date(DatesString,format="%d/%m/%Y")
> Dates
[1] "1732-02-22" "1735-10-30" "1743-04-13" "1751-03-16"
> DatesPlusOne <- Dates + 1
> 
```

Write the new vector out by typing:

DatesPlusOne



Run the line of script to console:



It can be observed that a day has been subtracted from the Dates vector?

Procedure 6: Extract Reporting Periods from a Date.

There are many functions that exist to extract useful information from dates such as weekdays, months or quarters which make reporting on dates more native. This procedure focusses on three functions:

- weekdays() which extracts the particular day of the week (e.g. Monday).
- months() which extracts the month of the year (e.g. June).
- quarters() which extracts the quarter of the date in the year (e.g. Q3).

All these functions work in the same manner, in that they take just one date argument and return a value. In this example, the quarter is to be returned for the purpose of reporting. To return the quarter value:

ReportingQuarters <- quarters(Dates)

```
Double <- c(1.22341,5.889988,6.9999890)
DoubleRound <- round(Double,2)
DoubleRound
DoubleRoundSqrt <- sqrt(DoubleRound)
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <- substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl("^Richard",Names)
NamesGrepl
NamesGreplNumeric <- as.numeric(NamesGrepl)
NamesGreplNumeric
NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
DatesString
Dates <- as.Date(DatesString,format="%d/%m/%Y")
Dates
DatesPlusOne <- Dates + 1
DatesPlusOne
ReportingQuarters <- quarters(Dates)
```

Run the line of script to console:

```
> NamesSubstrLogical
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> Dates <- as.Date(DatesString,format="%d/%m/%Y")
> Dates
[1] "1732-02-22" "1735-10-30" "1743-04-13" "1751-03-16"
> DatesPlusOne <- Dates + 1
> DatesPlusOne
[1] "1732-02-23" "1735-10-31" "1743-04-14" "1751-03-17"
> ReportingQuarters <- quarters(Dates)
>
```

Writing out the vector typing:

ReportingQuarters

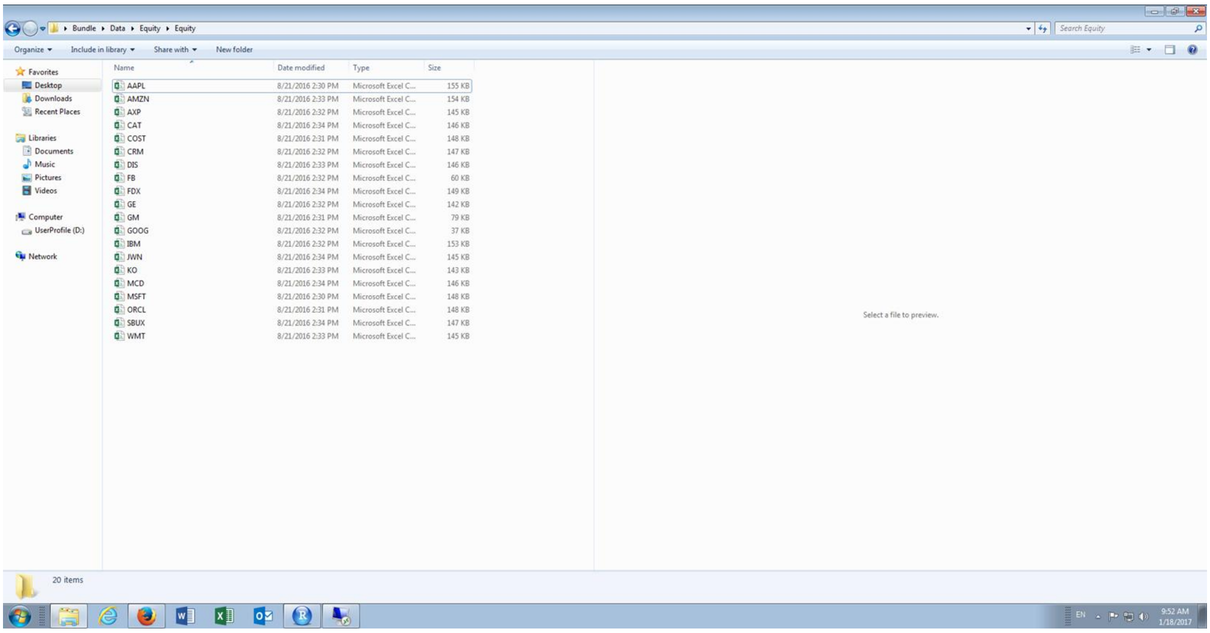
```
> NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> Dates <- as.Date(DatesString,format="%d/%m/%Y")
> Dates
[1] "1732-02-22" "1735-10-30" "1743-04-13" "1751-03-16"
> DatesPlusOne <- Dates + 1
> DatesPlusOne
[1] "1732-02-23" "1735-10-31" "1743-04-14" "1751-03-17"
> ReportingQuarters <- quarters(Dates)
> ReportingQuarters
[1] "Q1" "Q4" "Q2" "Q1"
>
```

It can be observed that the new vectors details the quarter extracted from the Dates() vector. The procedure may be used interchangeable between the weekdays() and months() function.

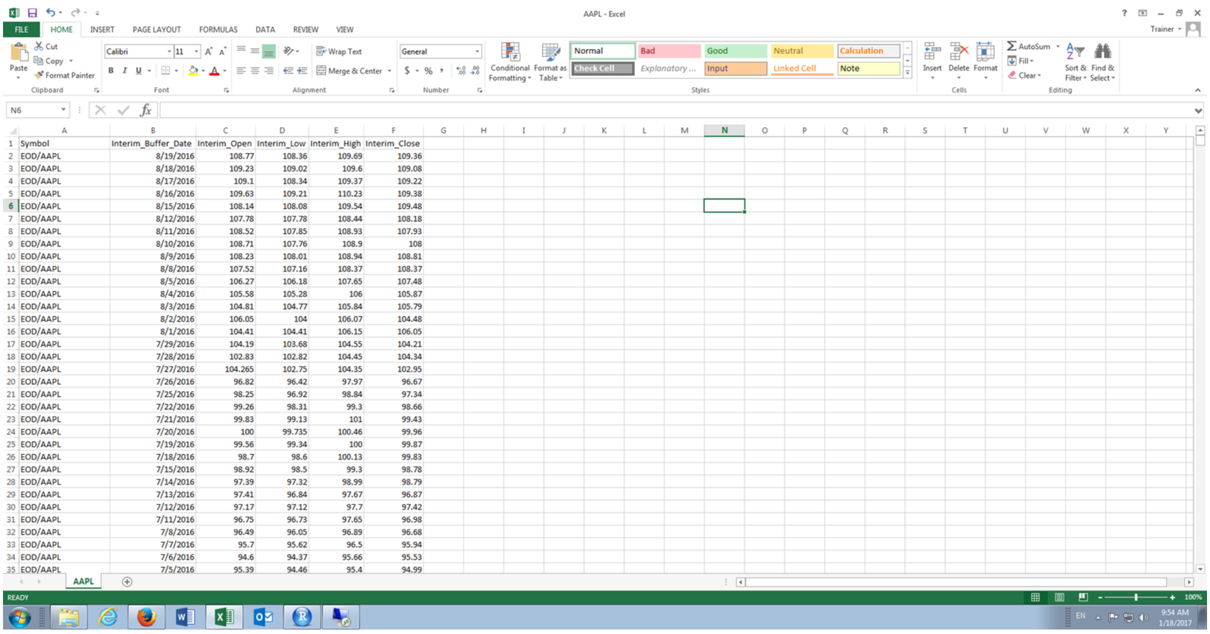
Procedure 7: Importing a CSV file with R Studio.

RStudio offers a simple GUI user interface to load files into Data Frames. The functionality is of course distinct to RStudio but in practice it is a code creator that uses the read.table() function to load a variety of common file formats to a Data Frame.

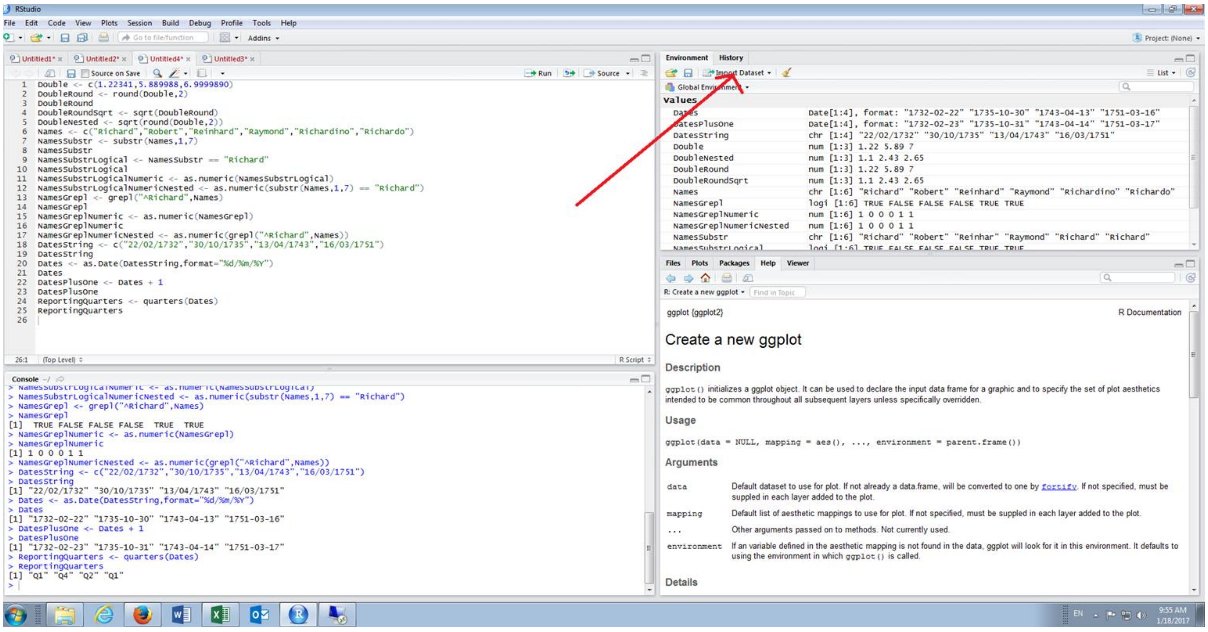
The procedure here in will use the datasets contained in the bundle. In this procedure, the csv datasets contained in \Bundle\Data\Equity\Equity will be targeted:



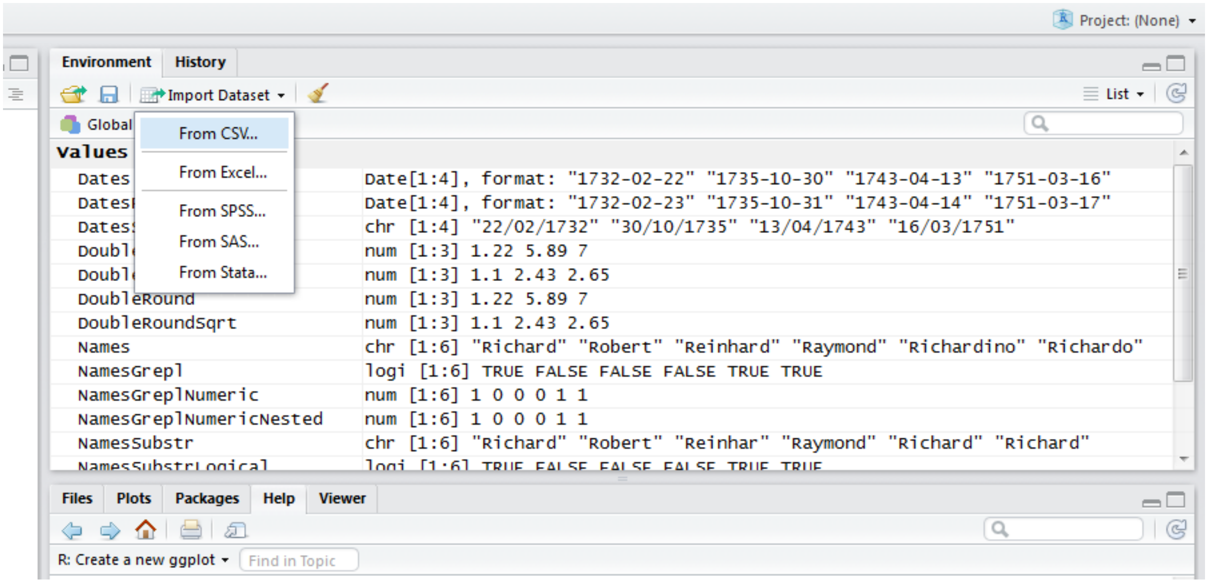
Specifically, the AAPL.csv file which contains a series of prices relating to the Apple share price:



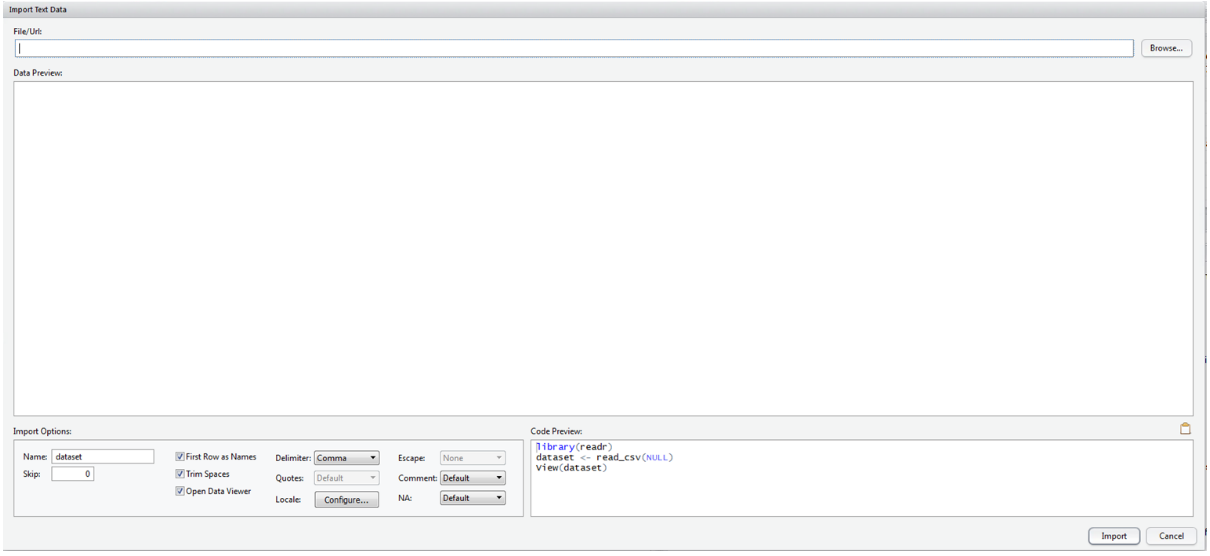
In RStudio, navigate to the Import Dataset button in the top right-hand corner of the screen, above the environment pane:



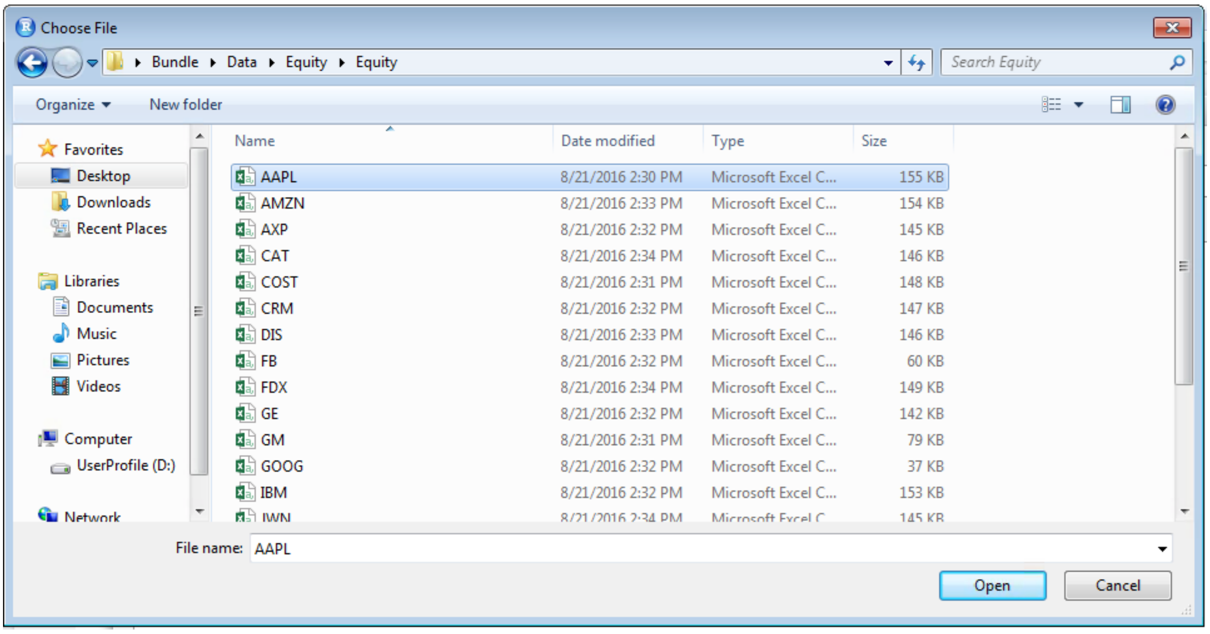
Click the button Import Dataset:



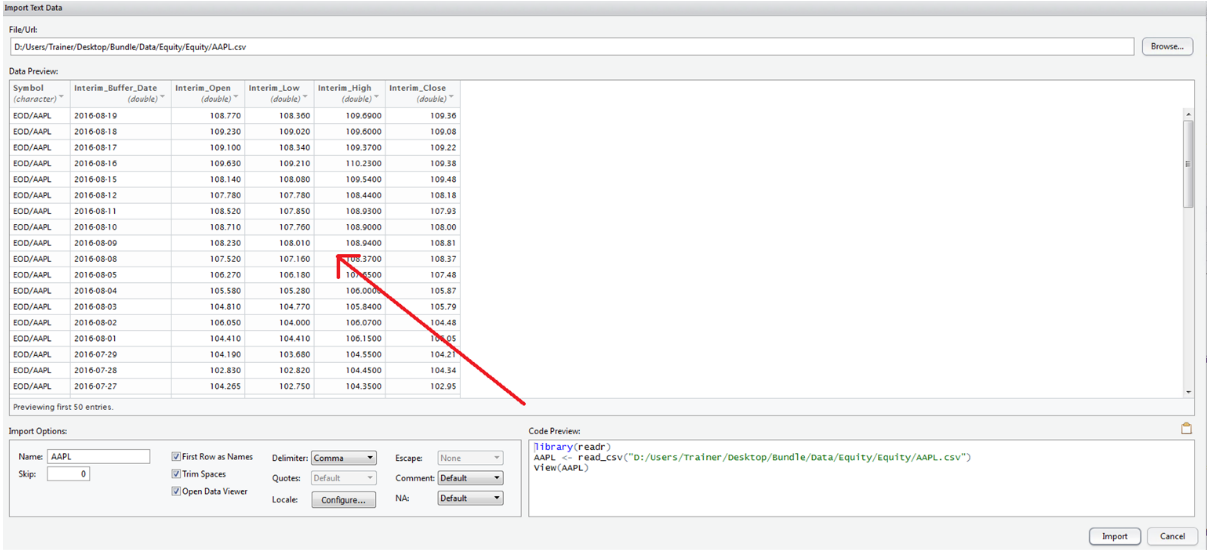
Click the From CVS sub menu:



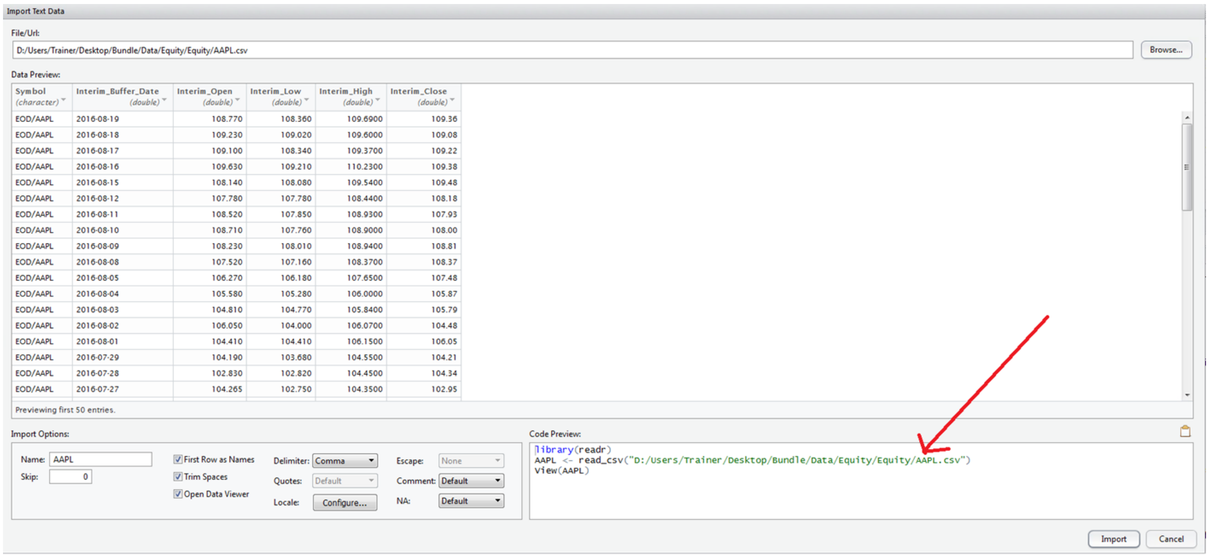
The Import Text file window will expand. Click the browse button in the top right-hand corner of the window to open the file system navigator:



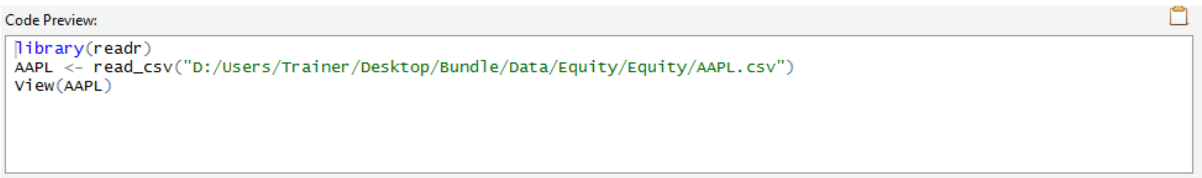
Navigate to Bundle\Data\Equity\Equity\AAPL.csv and click the Open button:



A preview of the file is show in the window for the purposes of validation:

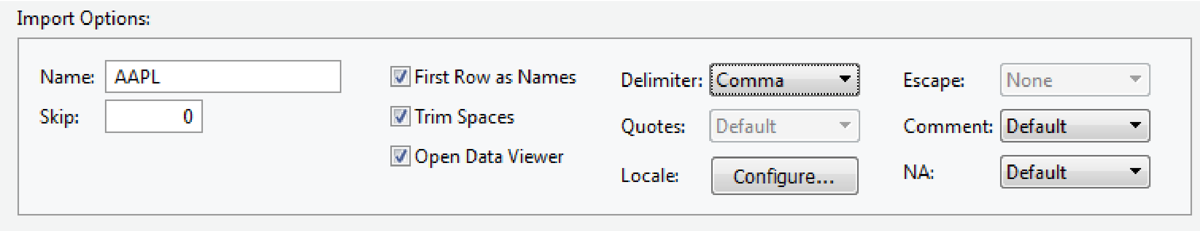


As is the case with many RStudio functions it is in essence a macro or code creation widget. It can be seen in the bottom right hand corner that RStudio has created the corresponding R script block that will be responsible for importing the file in the console:

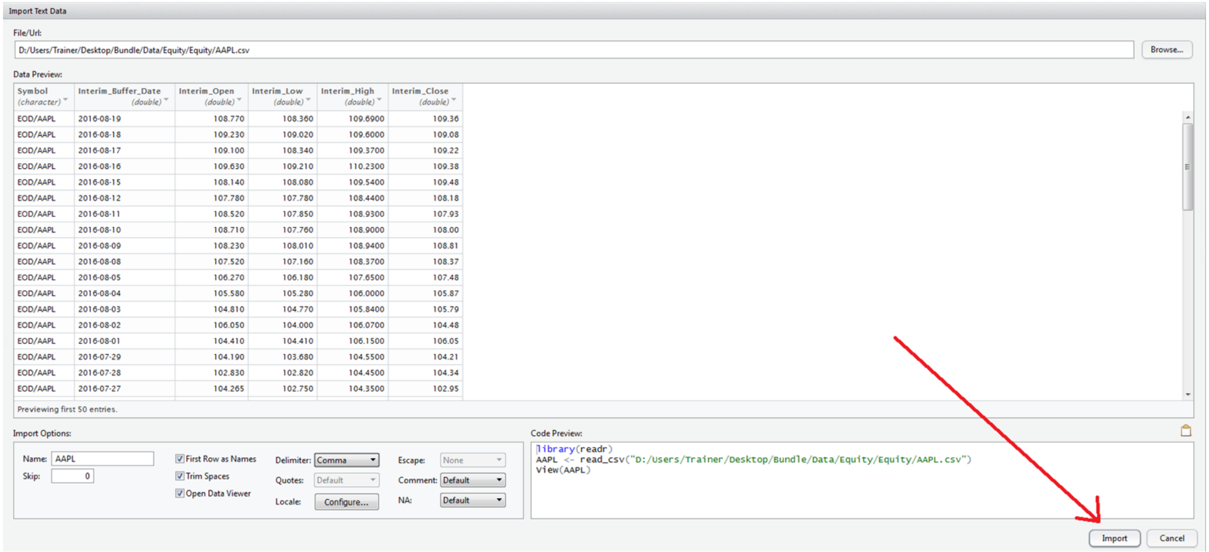


In this example, it can be observed that the readr package is being loaded, the csv file is being loaded to a data frame called AAPL using the read_csv function. The readr is a more efficient package for the importing and exporting of data created by the RStudio team and while there are several functions for the import and export of data native to R, these are not especially performant. It is worth noting that this package WILL NOT convert strings to factors, making it a more labour-intensive choice for text rich datasets that are intended to be the source of predictive analytics methods.

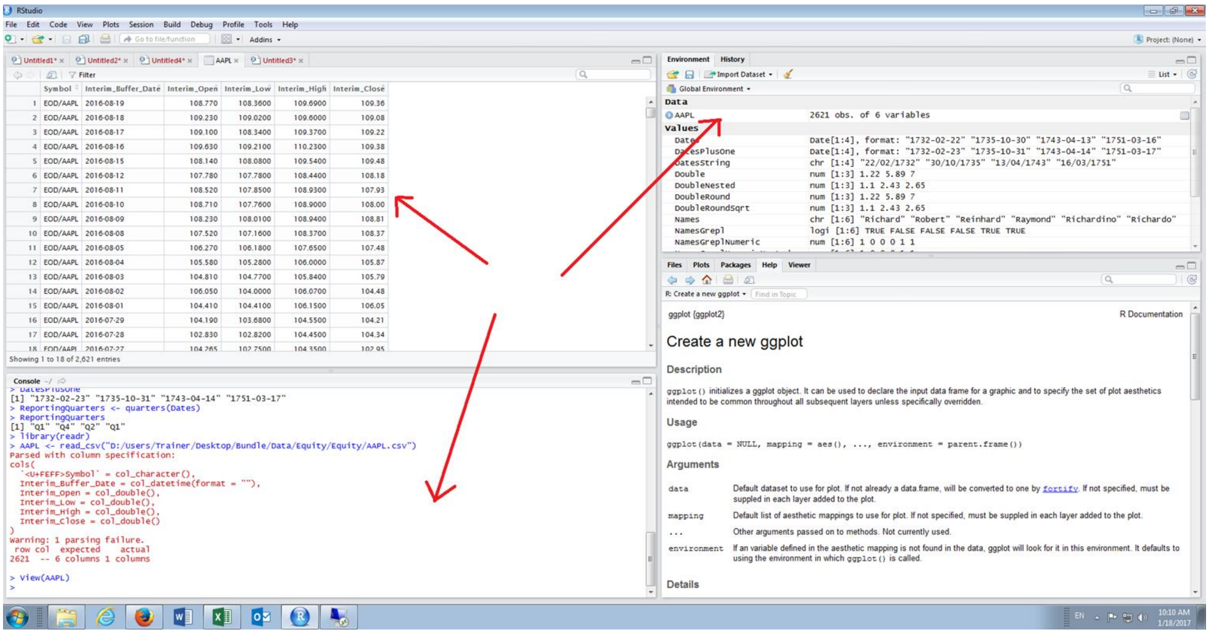
Towards the bottom left hand corner of window is additional parameters available in the creation of the csv file.



Simply click import to load the data into the R session:

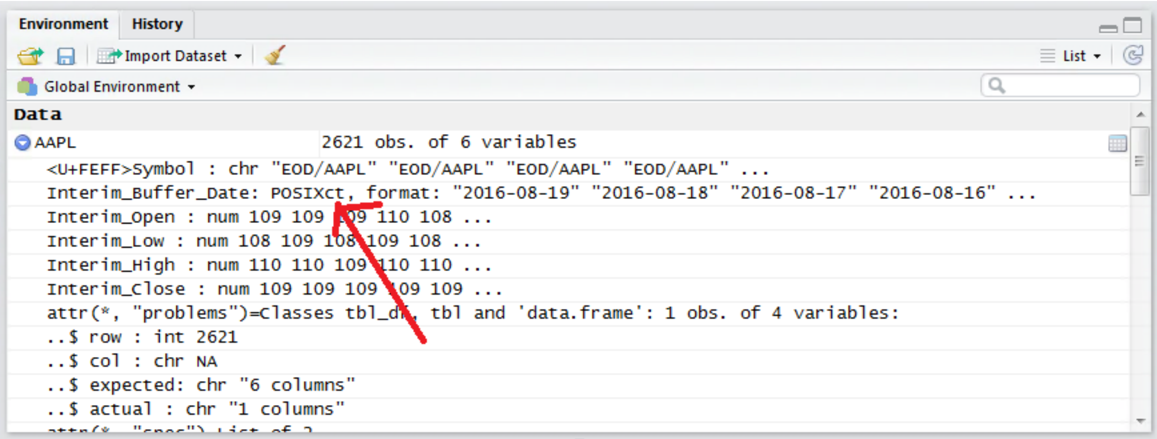


It can be seen that the block of script has been run to console, that the AAPL data frame is now available in the environment pane and care of the View() function, that the data frame has been displayed in a tab of the script pane:



It is important to note that all RStudio had done is create a block of R script and executed this to console. In the interests of reproducibility and in a script active console passive methodology, this block of script should be reproduced directly in a script. By way of standard, the readr package will be used in most, but not all, importing methods.

Expanding on the data frame it can be observed that the readr package has facilitated the creation of the correct object types:



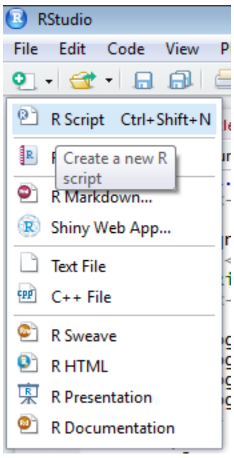
In this case, it can be seen that the handling of dates has taken place via POSIXCT, which is an alternative date handling object as detailed in procedure 43.

Procedure 8: Importing a pipe separated file.

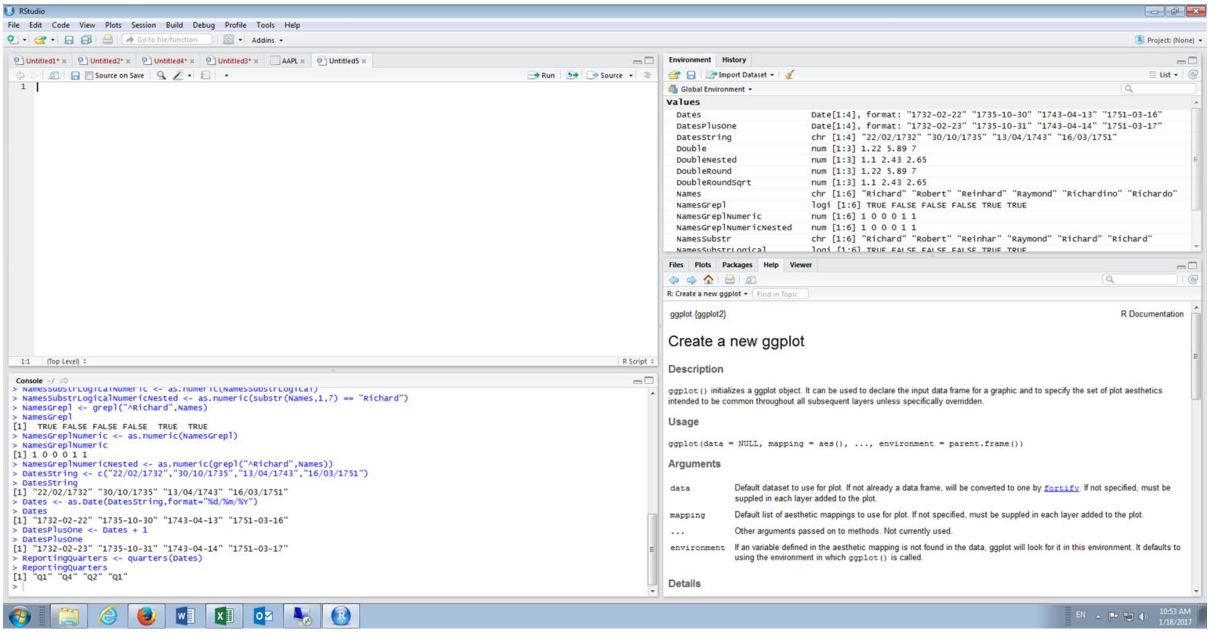
While a csv file is the most prolific means to exchange datasets, it is not by any means the only structure of text file. Other types of delimiter, this is to say using something other than a comma to separate the fields of a dataset, may include a pipe (i.e |) a tab, a semicolon (;) or just a space.

The readr package provides for the importing of data which has a slightly different structure to a csv file. This procedure will not use RStudio, instead focus on creating a script for the purposes of reproducibility.

Create a new script window in RStudio by navigating to clicking on the new script icon, then clicking RScript:

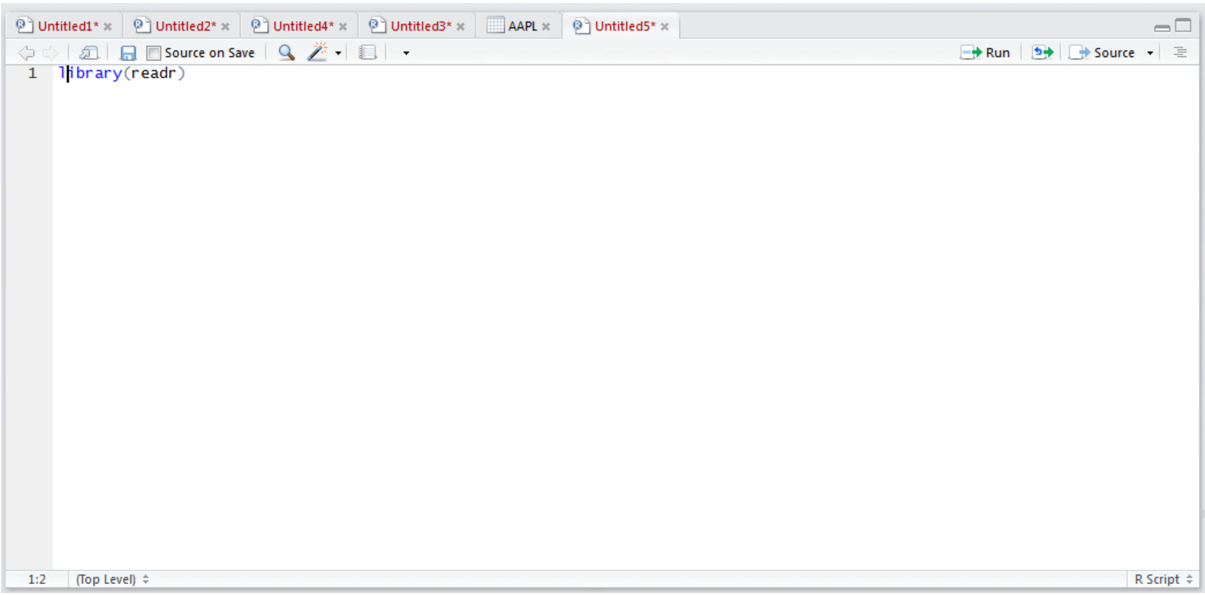


A blank script will be created:



Start by loading the readr library by typing:

library(readr)

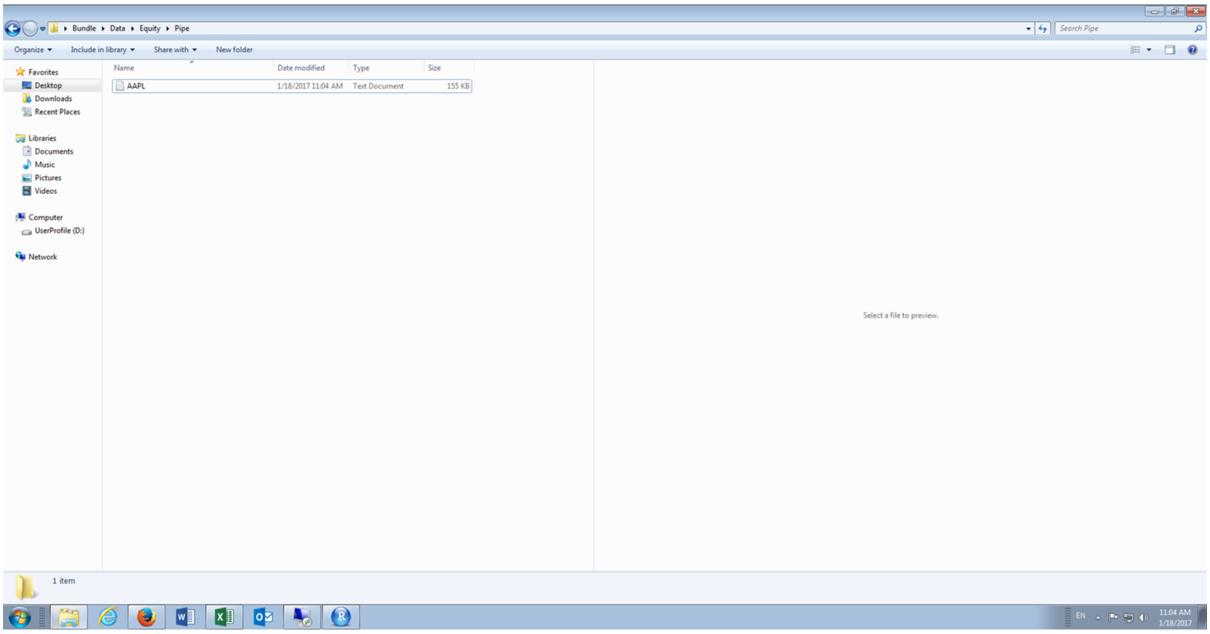


Run the line of script to console:

JUBE

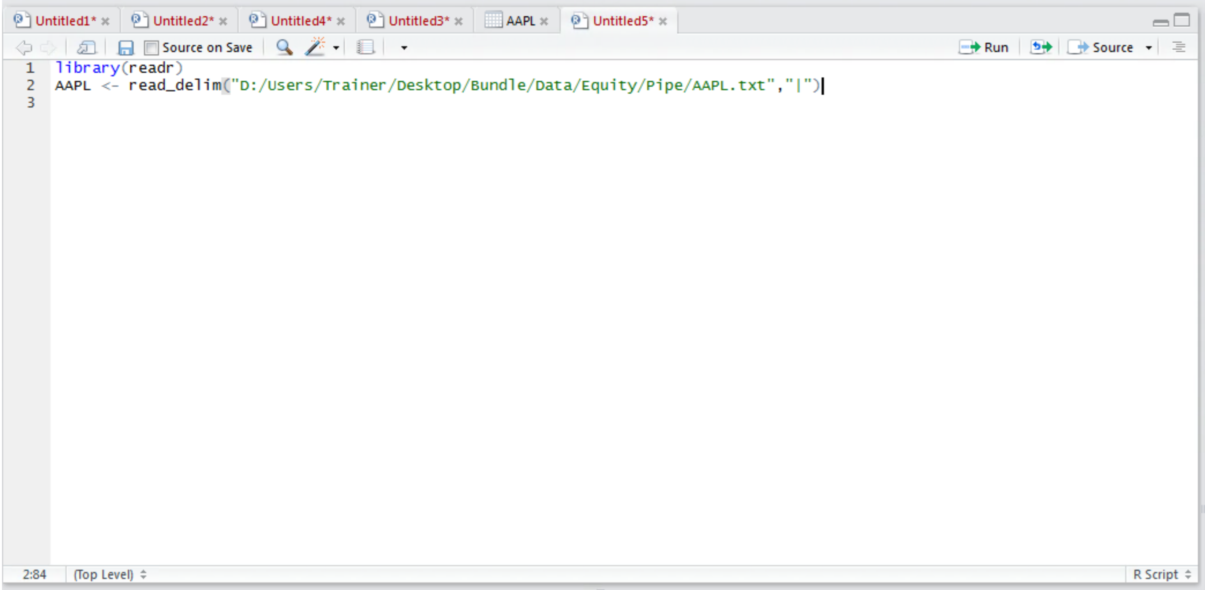
```
Console ~/
> NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
> NamesGrepl <- grepl("^Richard",Names)
> NamesGrepl
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
> NamesGreplNumeric <- as.numeric(NamesGrepl)
> NamesGreplNumeric
[1] 1 0 0 0 1 1
> NamesGreplNumericNested <- as.numeric(grepl("^Richard",Names))
> DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
> DatesString
[1] "22/02/1732" "30/10/1735" "13/04/1743" "16/03/1751"
> Dates <- as.Date(DatesString,format="%d/%m/%Y")
> Dates
[1] "1732-02-22" "1735-10-30" "1743-04-13" "1751-03-16"
> DatesPlusOne <- Dates + 1
> DatesPlusOne
[1] "1732-02-23" "1735-10-31" "1743-04-14" "1751-03-17"
> ReportingQuarters <- quarters(Dates)
> ReportingQuarters
[1] "Q1" "Q4" "Q2" "Q1"
> library(readr)
> |
```

In this example, a file containing the same data as imported in procedure 46 will be used albeit the delimiter is a pipe and not a comma. The file is available in Bundle\Data\Equity\Pipe\AAPL.txt:

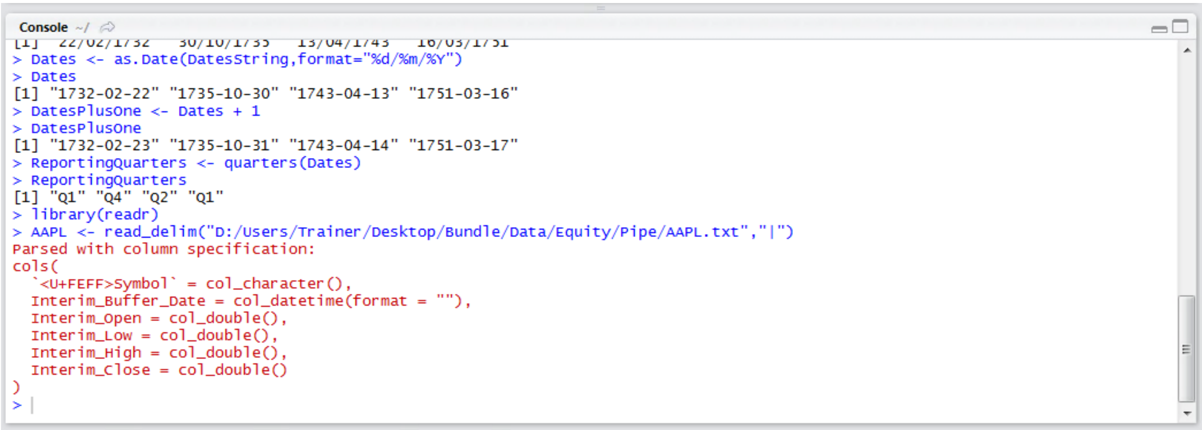


To import the pipe delimited file use the read_delim() function of the readr package. The function takes the arguments of the name and location of the file (in this case Bundle\Data\Equity\Pipe\AAPL.txt) then the delimiter (in this case |). To layout the read_delim() function type:

AAPL <- Read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")

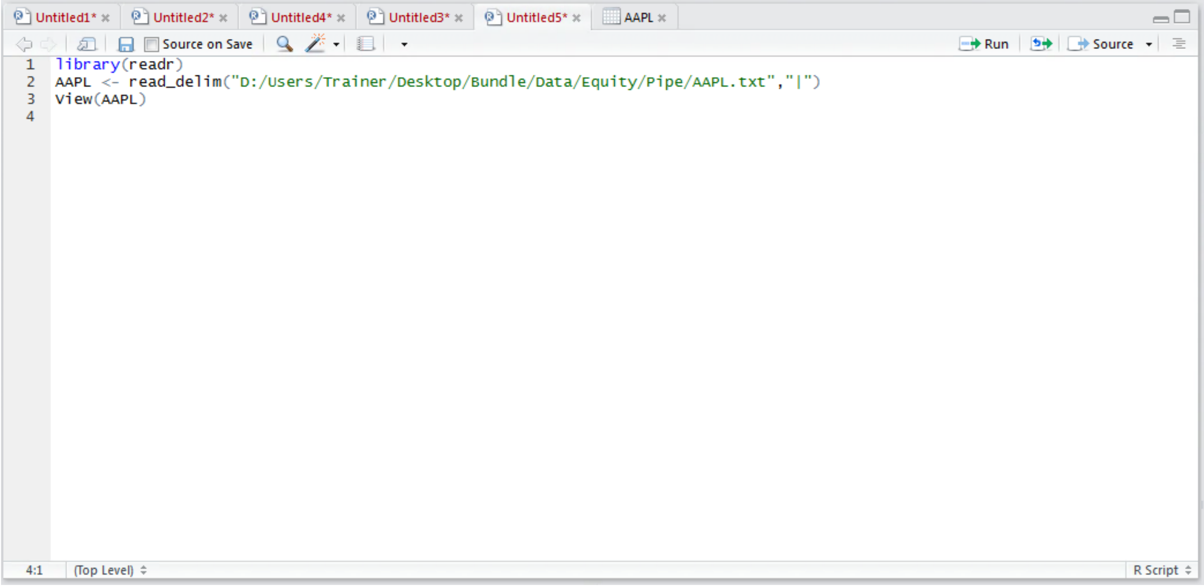


Note that the default backslash file structure used in windows (i.e. \) has been changed to a forward slash (i.e. /). Further in this example it is important to change the preceding file location of the bundle to the correct location on the computer (i.e. D:/Users/Trainer/Desktop/). Run the line of script to console:

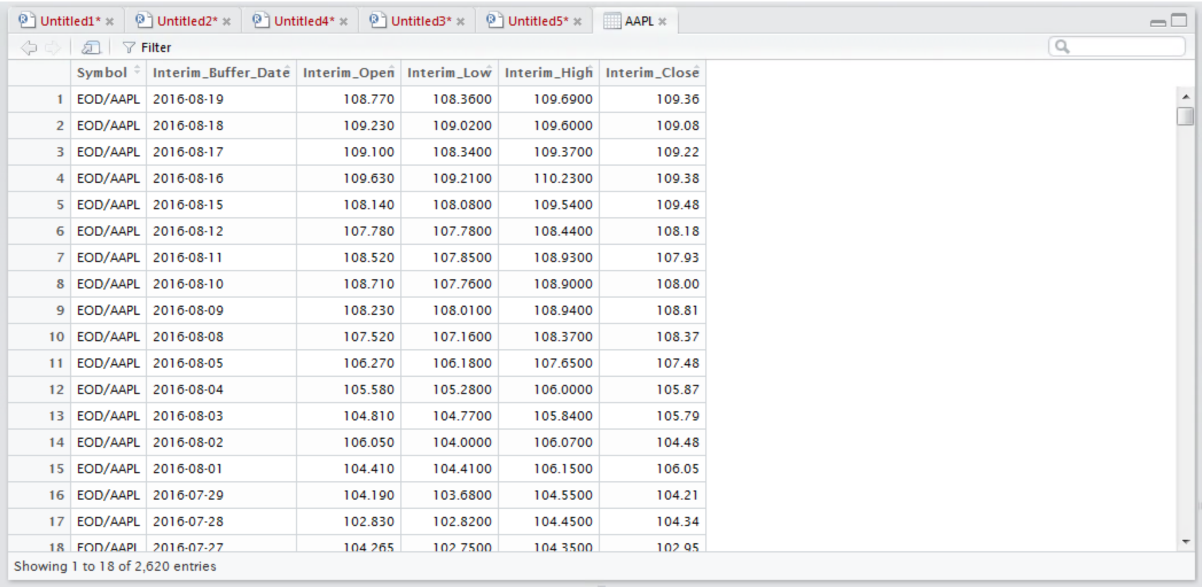


It can be seen that the specification for the data frame has been written out and that there are now errors. View, and validate, the import by typing:

View(AAPL)



Run the line of script to console to expand the data frame to the script window:



	Symbol	Interim_Buffer_Date	Interim_Open	Interim_Low	Interim_High	Interim_Close
1	EOD/AAPL	2016-08-19	108.770	108.3600	109.6900	109.36
2	EOD/AAPL	2016-08-18	109.230	109.0200	109.6000	109.08
3	EOD/AAPL	2016-08-17	109.100	108.3400	109.3700	109.22
4	EOD/AAPL	2016-08-16	109.630	109.2100	110.2300	109.38
5	EOD/AAPL	2016-08-15	108.140	108.0800	109.5400	109.48
6	EOD/AAPL	2016-08-12	107.780	107.7800	108.4400	108.18
7	EOD/AAPL	2016-08-11	108.520	107.8500	108.9300	107.93
8	EOD/AAPL	2016-08-10	108.710	107.7600	108.9000	108.00
9	EOD/AAPL	2016-08-09	108.230	108.0100	108.9400	108.81
10	EOD/AAPL	2016-08-08	107.520	107.1600	108.3700	108.37
11	EOD/AAPL	2016-08-05	106.270	106.1800	107.6500	107.48
12	EOD/AAPL	2016-08-04	105.580	105.2800	106.0000	105.87
13	EOD/AAPL	2016-08-03	104.810	104.7700	105.8400	105.79
14	EOD/AAPL	2016-08-02	106.050	104.0000	106.0700	104.48
15	EOD/AAPL	2016-08-01	104.410	104.4100	106.1500	106.05
16	EOD/AAPL	2016-07-29	104.190	103.6800	104.5500	104.21
17	EOD/AAPL	2016-07-28	102.830	102.8200	104.4500	104.34
18	EOD/AAPL	2016-07-27	104.265	102.7500	104.3500	102.95

Showing 1 to 18 of 2,620 entries

Procedure 9: Connect to an SQL Server Database.

This training course has a module dedicated to the creation of SQL statements for data mining and wrangling, for the purposes of this procedure it is only necessary to introduce SQL Server as a relational database management platform comprised of tables which are little more than a static equivalent to a csv file.

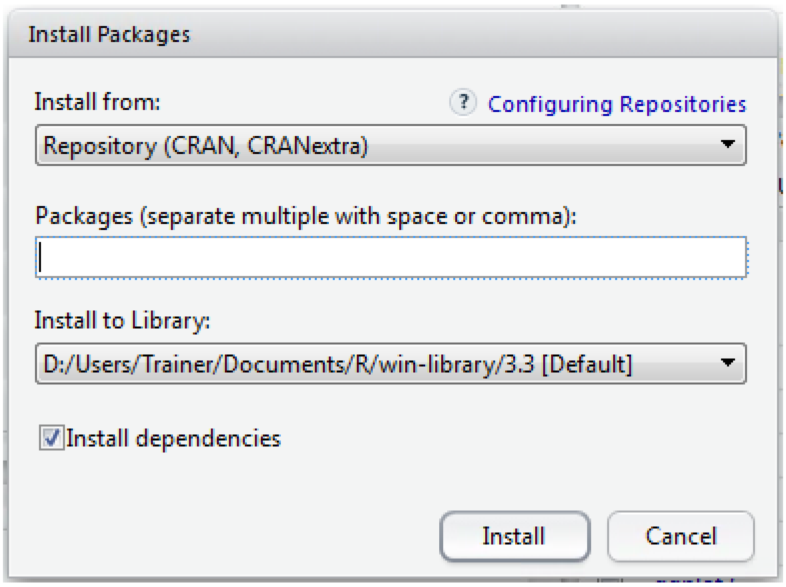
To connect to an SQL Server, the first step is to obtain the location of the server, the database name and credentials to log into this database, which for this document are detailed in the following table:

Credentials	String
Server	(local)/SQLEXPRESS
Database	Training
User	Sa

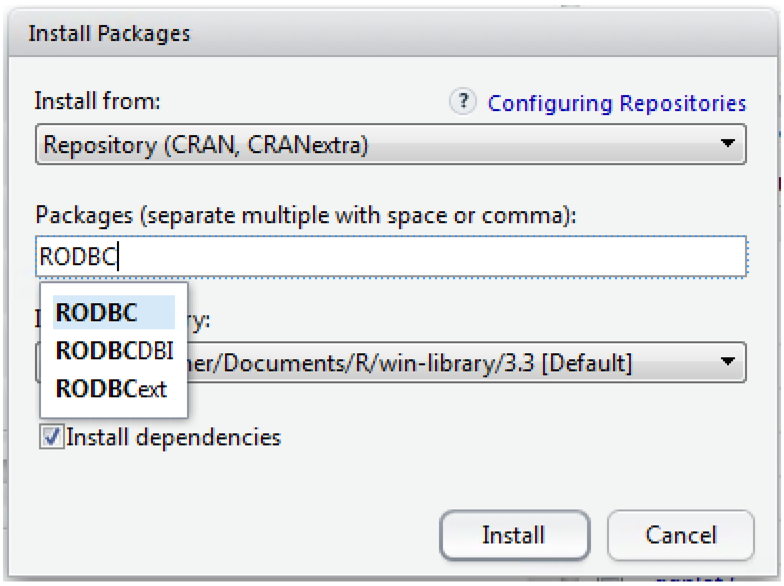
Password	Training12345

There are many different packages that facilitate the connection to databases for the purposes of retrieving tables and executing SQL. In this procedure, the RODBC (R Open Database Connectivity) will be used as it one of the most established packages available for the purposes of cross platform database connection.

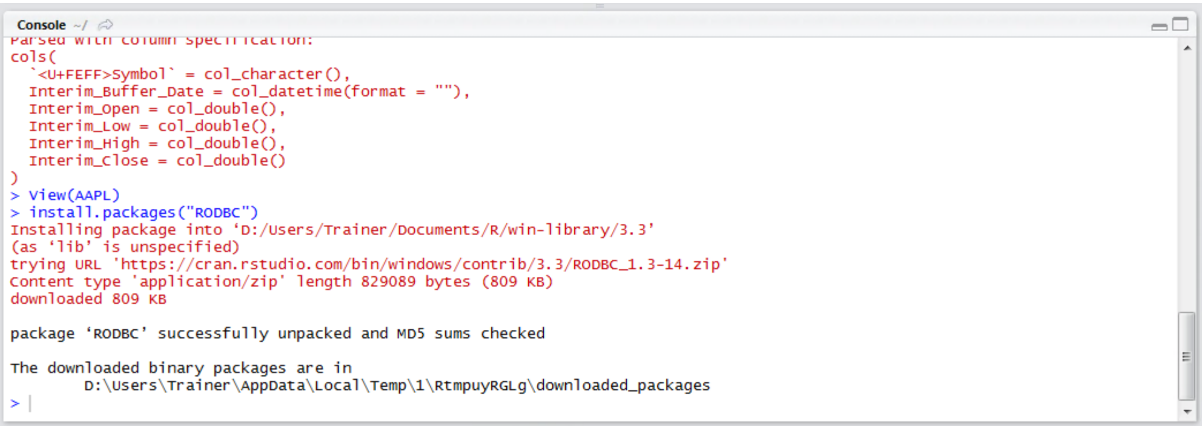
Firstly, RODBC relies on the RODBC package and as such this needs to be installed. Navigate to and click the install packaged button as per procedure 9:



The packages textbox will auto complete on the submission of the package name, in this case RODBC:

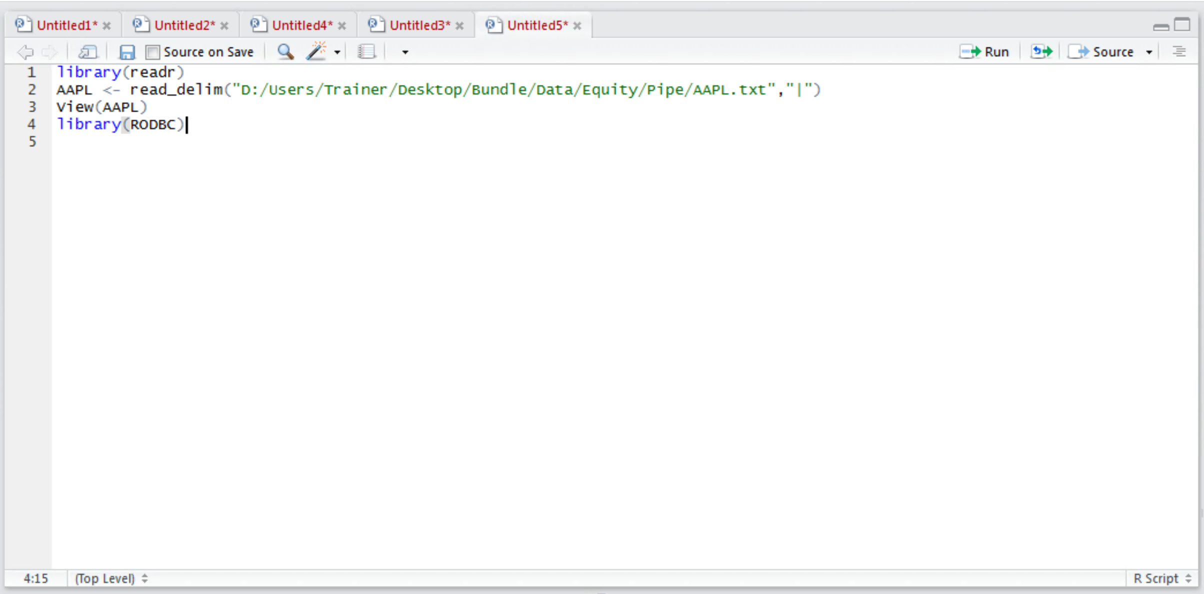


Click install to being the download and installation of the RODBC package:

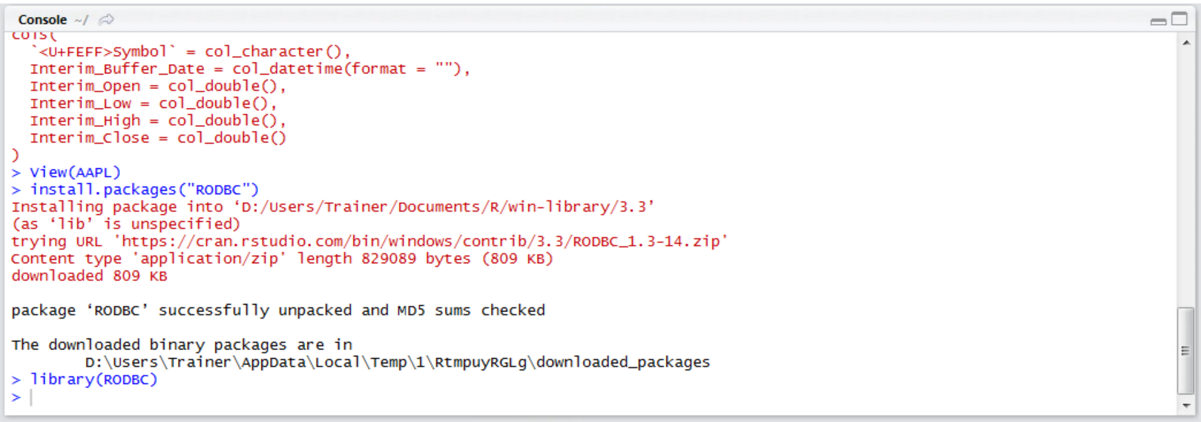


The package can be observed as having been installed, which will allow for the package to be referenced using the library() function. Navigate to the script pane and type:

library(RODBC)

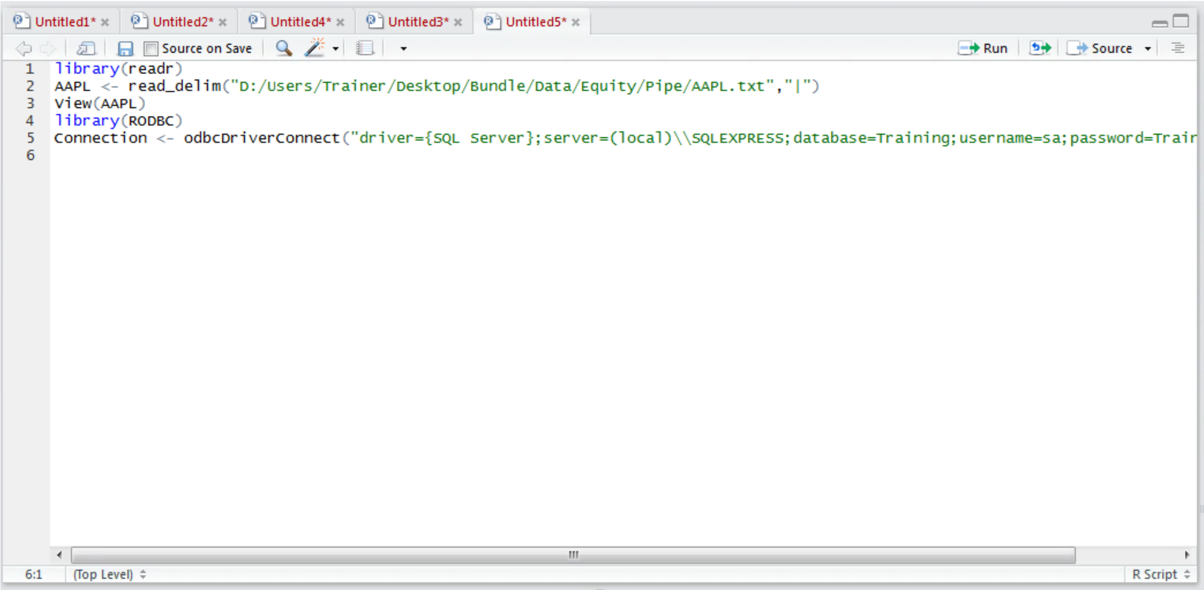


Run the line of script to console:



Databases maintain a static connection that should be explicitly opened and closed with the credentials of the database. To connect to an SQL Server database, retaining the connection for future use, type:

Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Training12345")



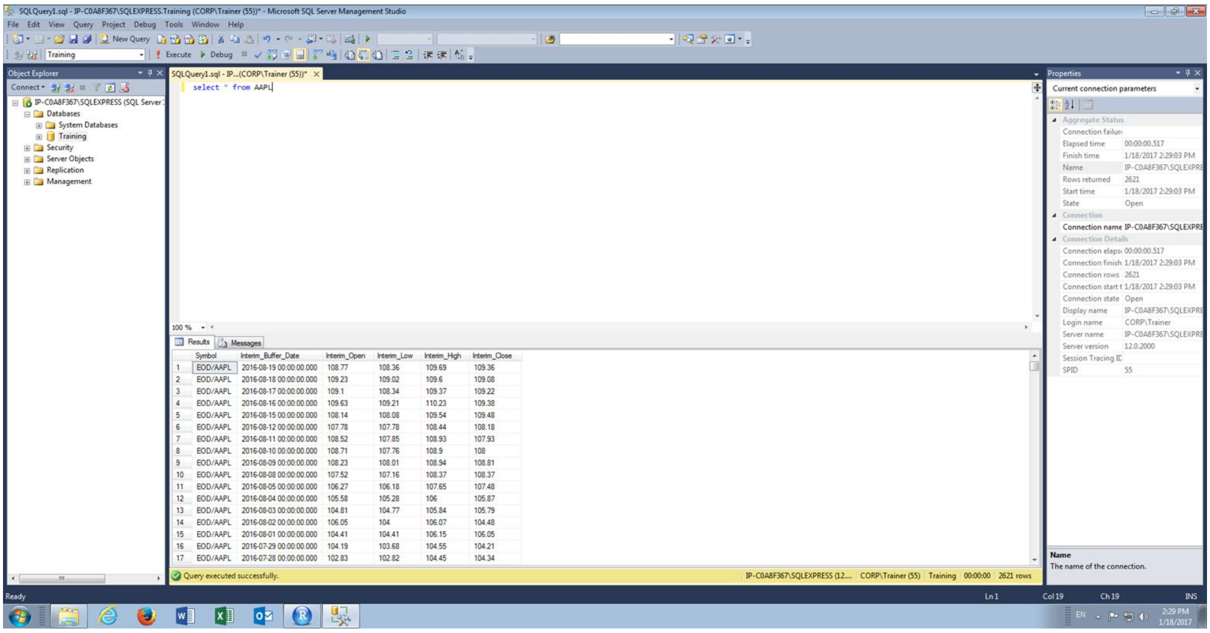
Notice how a backslash has special meaning in R, hence it has been escaped with a double backslash.

Run the line of script to console:

The absence of any errors is a signal that the connection to the database has been established successfully.

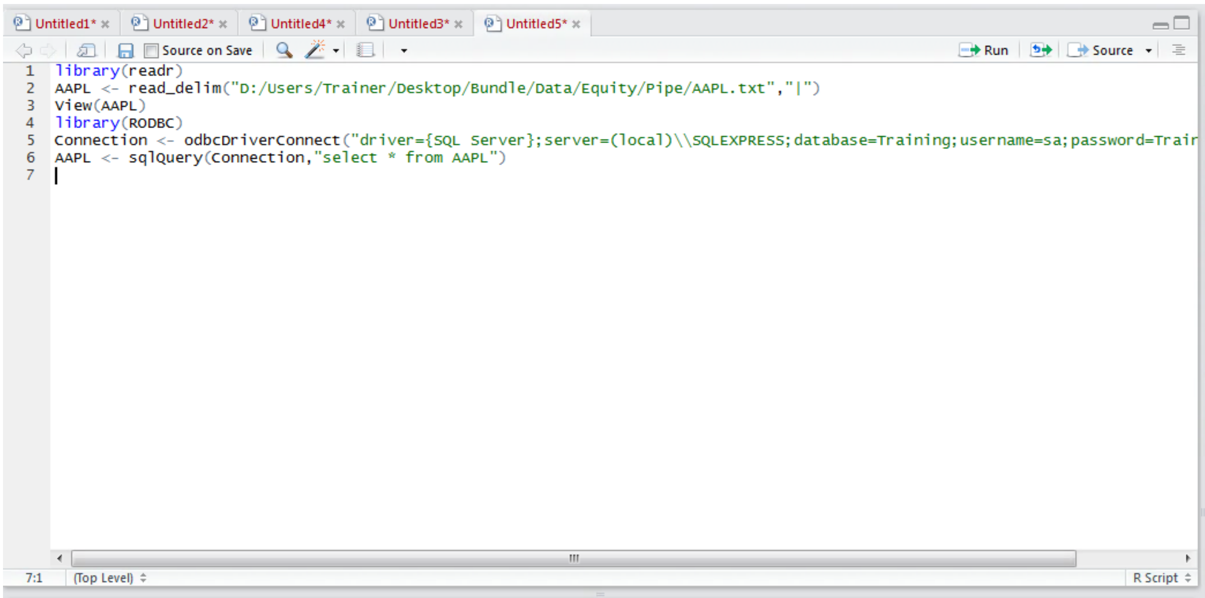
Procedure 10: Fetch an entire table from an SQL Server Database.

It suffices, for the purpose of this procedure, that there is a table in the SQL Server database titled AAPL containing the same information as the AAPL.csv and AAPL.txt files loaded in procedure 46 and y:



Offloading data mining and wrangling to SQL Server is covered in much more detail in module 5. For the purposes of this procedure, select the contents of the table to a Data Frame by typing:

AAPL <- sqlQuery(Connection,"select * from AAPL")



Run the line of script to console to execute the SQL statement "select * from AAPL" via the connect established in procedure 48:

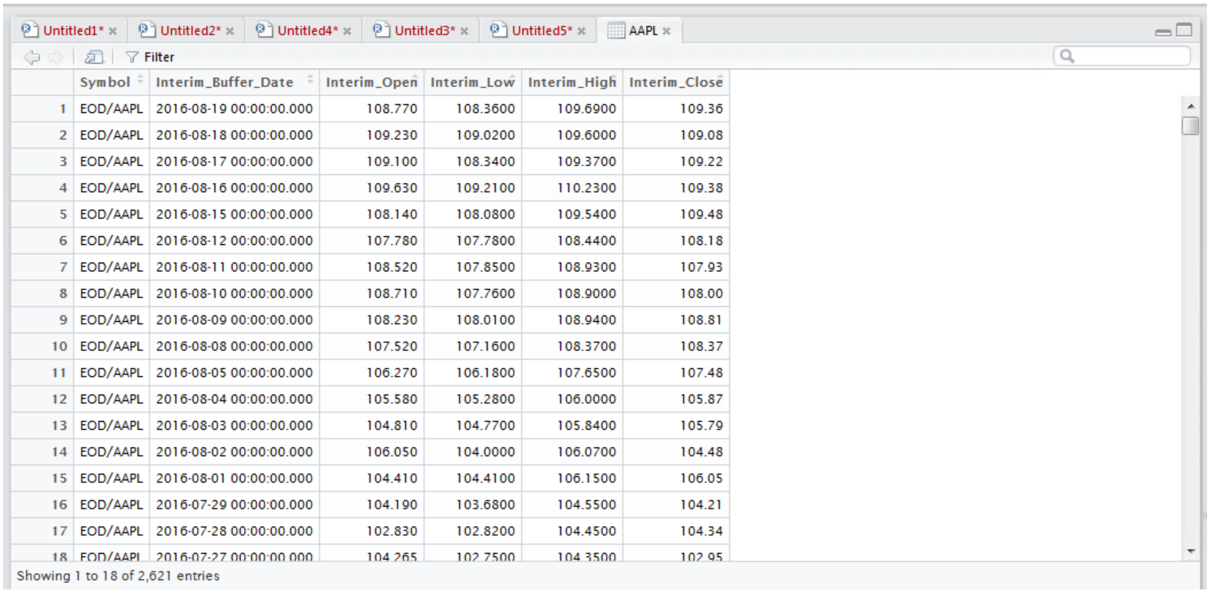
```
Console ~/
> DatesPlusOne
[1] "1732-02-23" "1735-10-31" "1743-04-14" "1751-03-17"
> ReportingQuarters <- quarters(Dates)
> ReportingQuarters
[1] "Q1" "Q4" "Q2" "Q1"
> library(readr)
> AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
Parsed with column specification:
cols(
  `<U+FEFF>Symbol` = col_character(),
  Interim_Buffer_Date = col_datetime(format = ""),
  Interim_Open = col_double(),
  Interim_Low = col_double(),
  Interim_High = col_double(),
  Interim_Close = col_double()
)
> View(AAPL)
> library(RODBC)
> Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Train
ing12345")
> AAPL <- sqlQuery(Connection,"select * from AAPL")
> 
```

The absence of any errors indicates that the SQL Query ran successfully, while an execution of the View() function against the data frame can further offer validation:

View(AAPL)

```
Untitled1* Untitled2* Untitled4* Untitled3* Untitled5*
Source on Save                                  Run   Source
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trair
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
7:11   (Top Level)                                R Script
```

Run the line of script to console to expand the AAPL data frame into a table in the script section of RStudio:

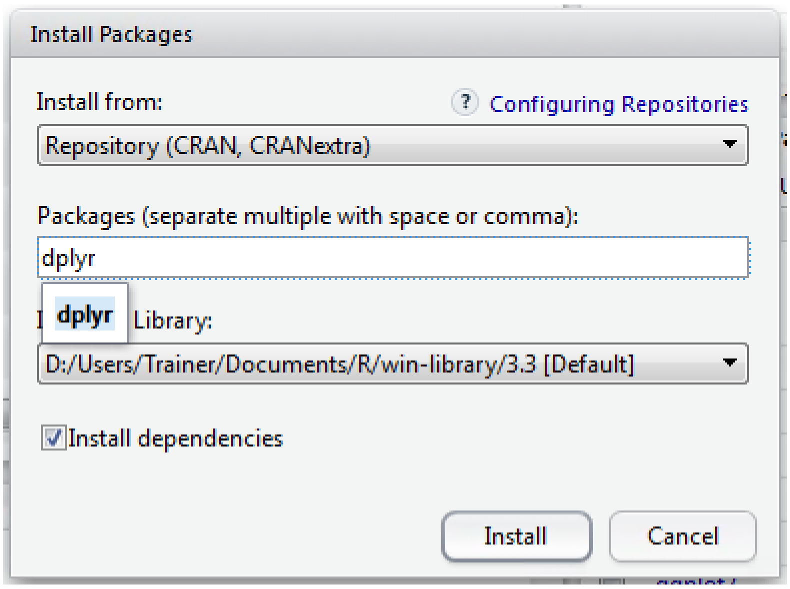


	Symbol	Interim_Buffer_Date	Interim_Open	Interim_Low	Interim_High	Interim_Close
1	EOD/AAPL	2016-08-19 00:00:00.000	108.770	108.3600	109.6900	109.36
2	EOD/AAPL	2016-08-18 00:00:00.000	109.230	109.0200	109.6000	109.08
3	EOD/AAPL	2016-08-17 00:00:00.000	109.100	108.3400	109.3700	109.22
4	EOD/AAPL	2016-08-16 00:00:00.000	109.630	109.2100	110.2300	109.38
5	EOD/AAPL	2016-08-15 00:00:00.000	108.140	108.0800	109.5400	109.48
6	EOD/AAPL	2016-08-12 00:00:00.000	107.780	107.7800	108.4400	108.18
7	EOD/AAPL	2016-08-11 00:00:00.000	108.520	107.8500	108.9300	107.93
8	EOD/AAPL	2016-08-10 00:00:00.000	108.710	107.7600	108.9000	108.00
9	EOD/AAPL	2016-08-09 00:00:00.000	108.230	108.0100	108.9400	108.81
10	EOD/AAPL	2016-08-08 00:00:00.000	107.520	107.1600	108.3700	108.37
11	EOD/AAPL	2016-08-05 00:00:00.000	106.270	106.1800	107.6500	107.48
12	EOD/AAPL	2016-08-04 00:00:00.000	105.580	105.2800	106.0000	105.87
13	EOD/AAPL	2016-08-03 00:00:00.000	104.810	104.7700	105.8400	105.79
14	EOD/AAPL	2016-08-02 00:00:00.000	106.050	104.0000	106.0700	104.48
15	EOD/AAPL	2016-08-01 00:00:00.000	104.410	104.4100	106.1500	106.05
16	EOD/AAPL	2016-07-29 00:00:00.000	104.190	103.6800	104.5500	104.21
17	EOD/AAPL	2016-07-28 00:00:00.000	102.830	102.8200	104.4500	104.34
18	EOD/AAPL	2016-07-27 00:00:00.000	104.265	102.7500	104.3500	102.95

Showing 1 to 18 of 2,621 entries

Procedure 11: Sorting a Data Frame with the arrange() function.

The procedures that follows are born of the dplyr package which is a collection of functions that exist for the purpose of shaping and moulding data frames. The first step is to ensure that the dplyr package is available by installing it through the Install section of the packages pane and as descried in procedure 9. Search for dplyr:



Click Install to download and install the dplyr package:

```
Console ~/
ing12345")
> AAPL <- sqlQuery(Connection,"select * from AAPL")
> View(AAPL)
> install.packages("dplyr")
Installing package into 'D:/Users/Trainer/Documents/R/win-library/3.3'
(as 'lib' is unspecified)
also installing the dependency 'DBI'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/DBI_0.5-1.zip'
Content type 'application/zip' length 364574 bytes (356 KB)
downloaded 356 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/dplyr_0.5.0.zip'
Content type 'application/zip' length 2408686 bytes (2.3 MB)
downloaded 2.3 MB

package 'DBI' successfully unpacked and MD5 sums checked
package 'dplyr' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpQBrXLm\downloaded_packages
> |
```

Load the dplyr library by typing:

library(dplyr)

```
Console ~/
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/dplyr_0.5.0.zip'
Content type 'application/zip' length 2408686 bytes (2.3 MB)
downloaded 2.3 MB

package 'DBI' successfully unpacked and MD5 sums checked
package 'dplyr' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpQBrXLm\downloaded_packages
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> |
```

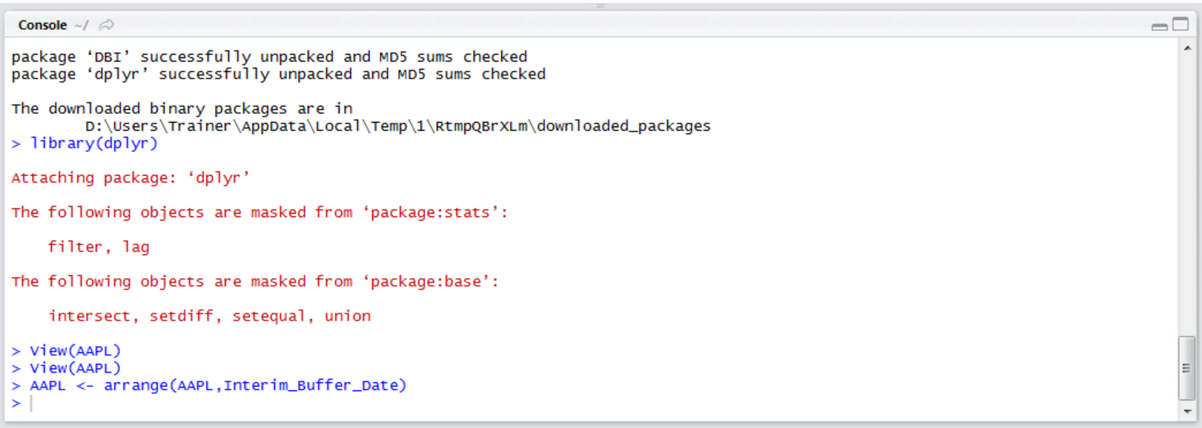
The package dplyr exposes several functions for shaping and moulding data. The arrange() function is used to rearrange, rather sort, the order of data in a data frame by columns in ascending order:

To arrange data by date for the AAPL data frame:

AAPL <- arrange(AAPL,Interim_Buffer_Date)

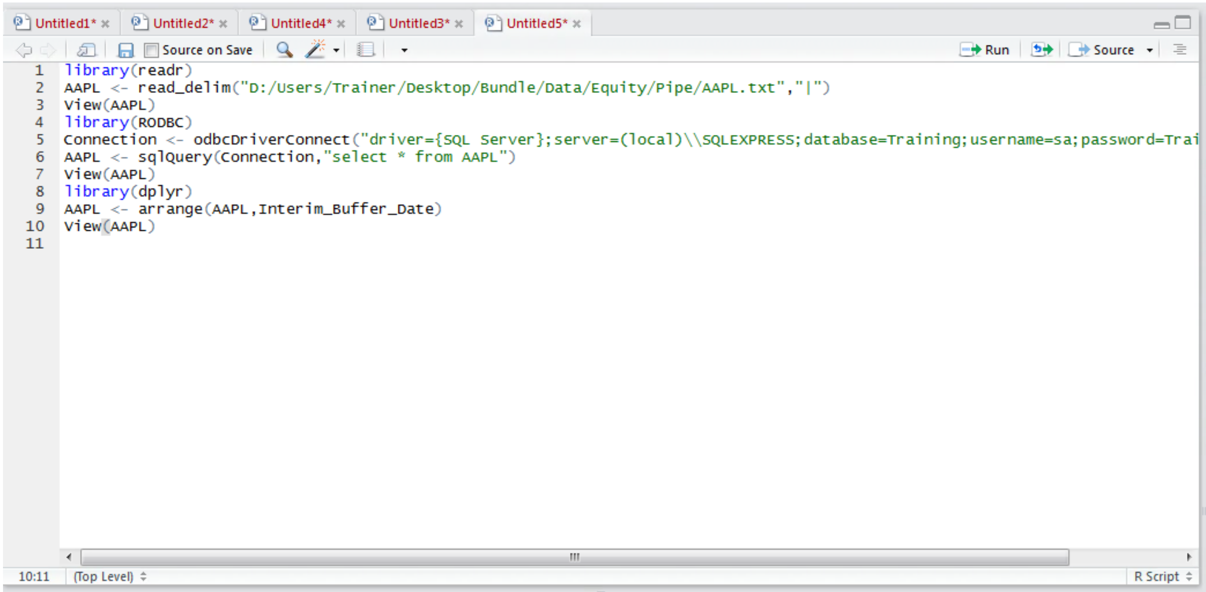
```
Untitled1* × Untitled2* × Untitled4* × Untitled3* × Untitled5* ×
Source on Save                                   Run   Source
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)

10:1  (Top Level) :                                                  R Script :
```

Run the line of script to console:

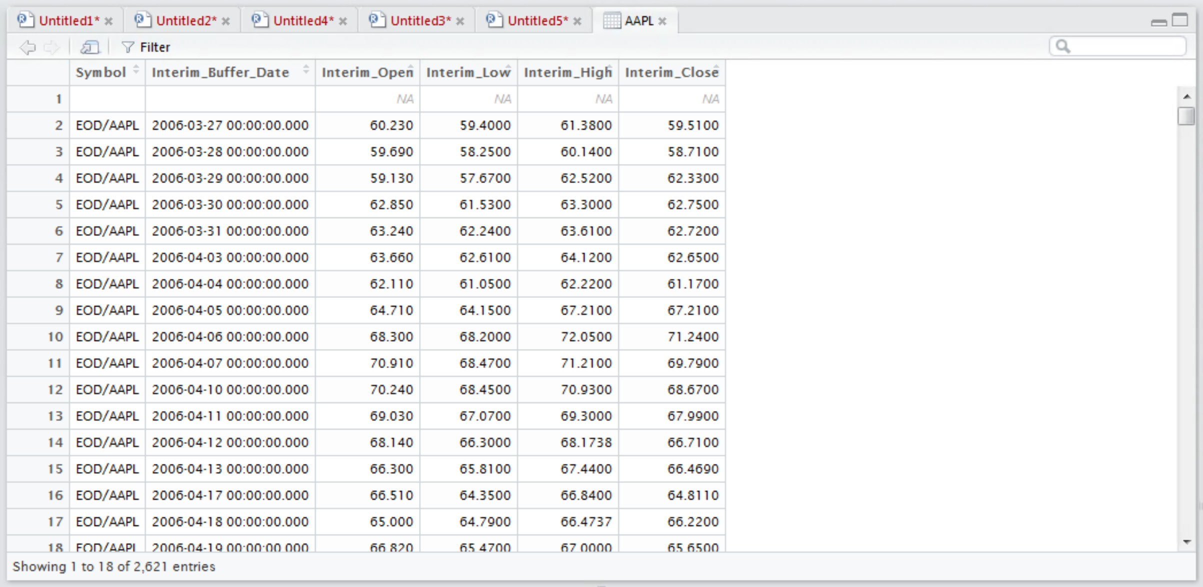


View the AAPL data frame to observe the change in row arrangement:

View(AAPL)



Run the line of script to console:

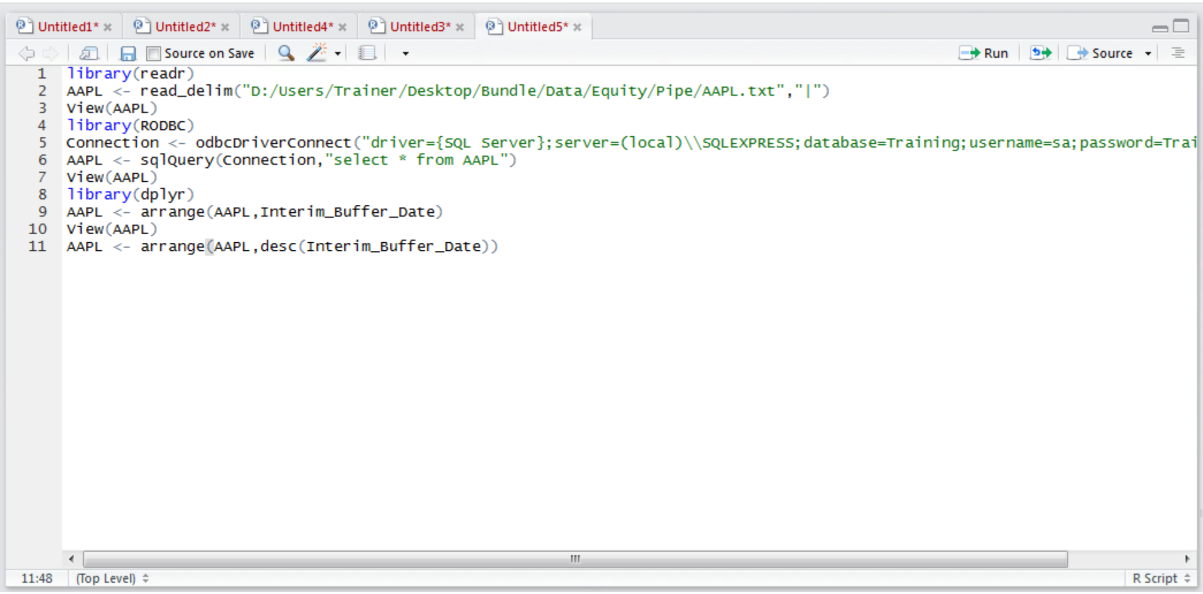


	Symbol	Interim_Buffer_Date	Interim_Open	Interim_Low	Interim_High	Interim_Close
1			NA	NA	NA	NA
2	EOD/AAPL	2006-03-27 00:00:00.000	60.230	59.4000	61.3800	59.5100
3	EOD/AAPL	2006-03-28 00:00:00.000	59.690	58.2500	60.1400	58.7100
4	EOD/AAPL	2006-03-29 00:00:00.000	59.130	57.6700	62.5200	62.3300
5	EOD/AAPL	2006-03-30 00:00:00.000	62.850	61.5300	63.3000	62.7500
6	EOD/AAPL	2006-03-31 00:00:00.000	63.240	62.2400	63.6100	62.7200
7	EOD/AAPL	2006-04-03 00:00:00.000	63.660	62.6100	64.1200	62.6500
8	EOD/AAPL	2006-04-04 00:00:00.000	62.110	61.0500	62.2200	61.1700
9	EOD/AAPL	2006-04-05 00:00:00.000	64.710	64.1500	67.2100	67.2100
10	EOD/AAPL	2006-04-06 00:00:00.000	68.300	68.2000	72.0500	71.2400
11	EOD/AAPL	2006-04-07 00:00:00.000	70.910	68.4700	71.2100	69.7900
12	EOD/AAPL	2006-04-10 00:00:00.000	70.240	68.4500	70.9300	68.6700
13	EOD/AAPL	2006-04-11 00:00:00.000	69.030	67.0700	69.3000	67.9900
14	EOD/AAPL	2006-04-12 00:00:00.000	68.140	66.3000	68.1738	66.7100
15	EOD/AAPL	2006-04-13 00:00:00.000	66.300	65.8100	67.4400	66.4690
16	EOD/AAPL	2006-04-17 00:00:00.000	66.510	64.3500	66.8400	64.8110
17	EOD/AAPL	2006-04-18 00:00:00.000	65.000	64.7900	66.4737	66.2200
18	EOD/AAPL	2006-04-19 00:00:00.000	66.820	65.4700	67.0000	65.6500

Showing 1 to 18 of 2,621 entries

Run sort in a different direction can be achieved using the desc() function wrapped around the column to be sorted. To change the direction of sort order on the Interim_Buffer_Date type:

AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))



```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
```

Run the line of script to console:

```
Console ~/
The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpQBrXLm\downloaded_packages
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> View(AAPL)
> View(AAPL)
> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> |
```

Observe the change in sort order:

View(AAPL)

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
```

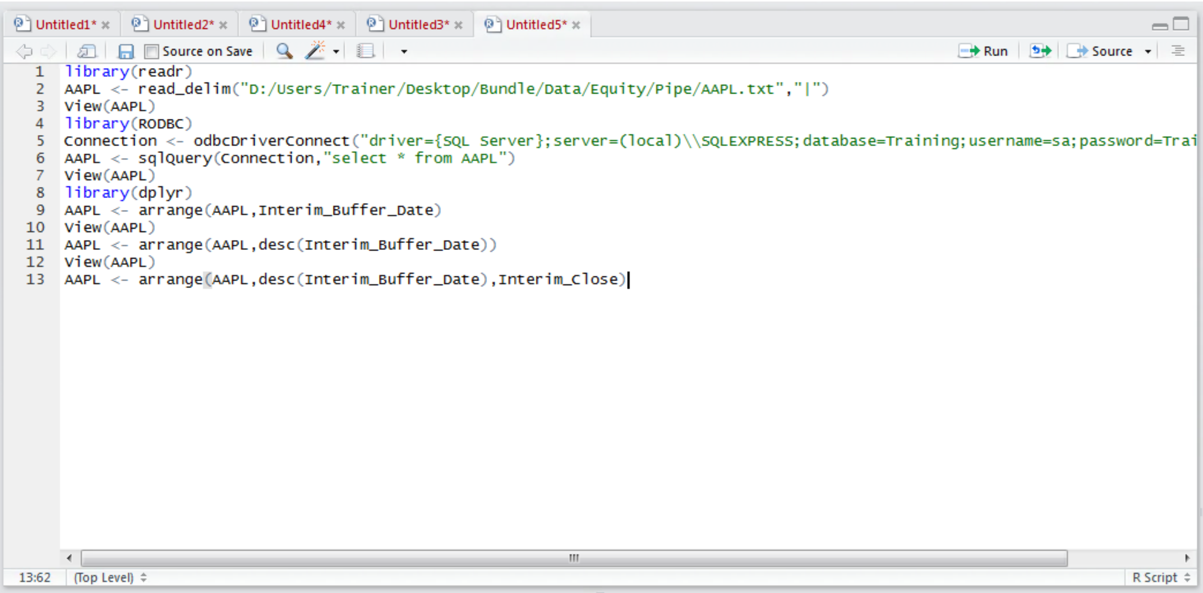
Run the line of script to console:

	Symbol	Interim_Buffer_Date	Interim_Open	Interim_Low	Interim_High	Interim_Close
1	EOD/AAPL	2016-08-19 00:00:00.000	108.770	108.3600	109.6900	109.36
2	EOD/AAPL	2016-08-18 00:00:00.000	109.230	109.0200	109.6000	109.08
3	EOD/AAPL	2016-08-17 00:00:00.000	109.100	108.3400	109.3700	109.22
4	EOD/AAPL	2016-08-16 00:00:00.000	109.630	109.2100	110.2300	109.38
5	EOD/AAPL	2016-08-15 00:00:00.000	108.140	108.0800	109.5400	109.48
6	EOD/AAPL	2016-08-12 00:00:00.000	107.780	107.7800	108.4400	108.18
7	EOD/AAPL	2016-08-11 00:00:00.000	108.520	107.8500	108.9300	107.93
8	EOD/AAPL	2016-08-10 00:00:00.000	108.710	107.7600	108.9000	108.00
9	EOD/AAPL	2016-08-09 00:00:00.000	108.230	108.0100	108.9400	108.81
10	EOD/AAPL	2016-08-08 00:00:00.000	107.520	107.1600	108.3700	108.37
11	EOD/AAPL	2016-08-05 00:00:00.000	106.270	106.1800	107.6500	107.48
12	EOD/AAPL	2016-08-04 00:00:00.000	105.580	105.2800	106.0000	105.87
13	EOD/AAPL	2016-08-03 00:00:00.000	104.810	104.7700	105.8400	105.79
14	EOD/AAPL	2016-08-02 00:00:00.000	106.050	104.0000	106.0700	104.48
15	EOD/AAPL	2016-08-01 00:00:00.000	104.410	104.4100	106.1500	106.05
16	EOD/AAPL	2016-07-29 00:00:00.000	104.190	103.6800	104.5500	104.21
17	EOD/AAPL	2016-07-28 00:00:00.000	102.830	102.8200	104.4500	104.34
18	EOD/AAPL	2016-07-27 00:00:00.000	104.265	102.7500	104.3500	102.95

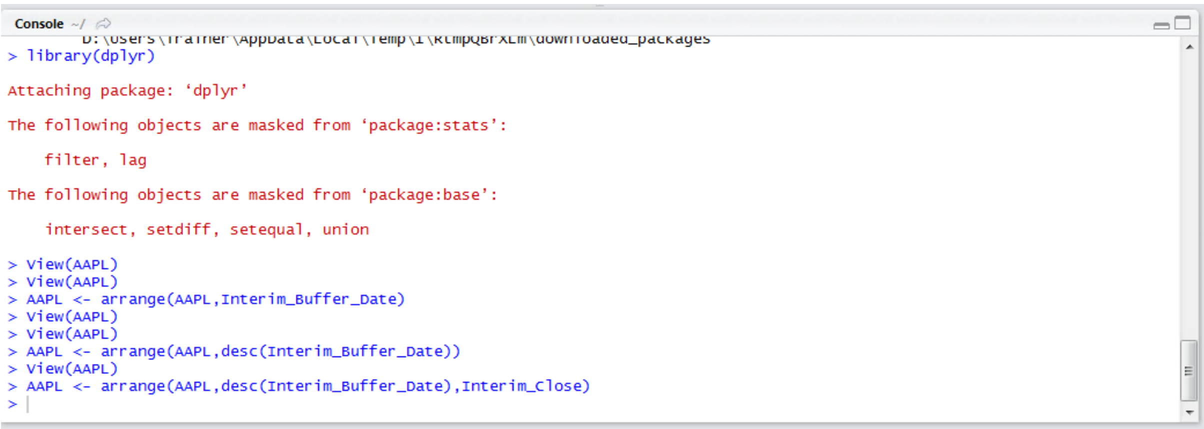
Showing 1 to 18 of 2,621 entries

It can be seen that the sort order has changed direction completely. To sort by one column, then the next, simply list out the columns in order then direction of the sort:

AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)



Run the line of script to console:



Procedure 12: Specifying columns of a Data Frame to return.

The select() function returns just the columns specified after the data frame. In this example, the AAPL data frame will be have some columns truncated leaving only the columns Interim_Buffer_Date and Interim_Close:

AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
```

Run the line of script to console:

```
ing12345")
> AAPL <- sqlQuery(Connection,"select * from AAPL")
> View(AAPL)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
> AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
>
```

View the data frame:

View(AAPL)

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
```

Run the line of script to console:

	Symbol	Interim_Buffer_Date	Interim_Close
1	EOD/AAPL	2016-08-19 00:00:00.000	109.36
2	EOD/AAPL	2016-08-18 00:00:00.000	109.08
3	EOD/AAPL	2016-08-17 00:00:00.000	109.22
4	EOD/AAPL	2016-08-16 00:00:00.000	109.38
5	EOD/AAPL	2016-08-15 00:00:00.000	109.48
6	EOD/AAPL	2016-08-12 00:00:00.000	108.18
7	EOD/AAPL	2016-08-11 00:00:00.000	107.93
8	EOD/AAPL	2016-08-10 00:00:00.000	108.00
9	EOD/AAPL	2016-08-09 00:00:00.000	108.81
10	EOD/AAPL	2016-08-08 00:00:00.000	108.37
11	EOD/AAPL	2016-08-05 00:00:00.000	107.48
12	EOD/AAPL	2016-08-04 00:00:00.000	105.87
13	EOD/AAPL	2016-08-03 00:00:00.000	105.79
14	EOD/AAPL	2016-08-02 00:00:00.000	104.48
15	EOD/AAPL	2016-08-01 00:00:00.000	106.05
16	EOD/AAPL	2016-07-29 00:00:00.000	104.21
17	EOD/AAPL	2016-07-28 00:00:00.000	104.34
18	EOD/AAPL	2016-07-27 00:00:00.000	102.95

Showing 1 to 18 of 2,621 entries

It can be observed that the data frame has discarded columns that were not specified explicitly.

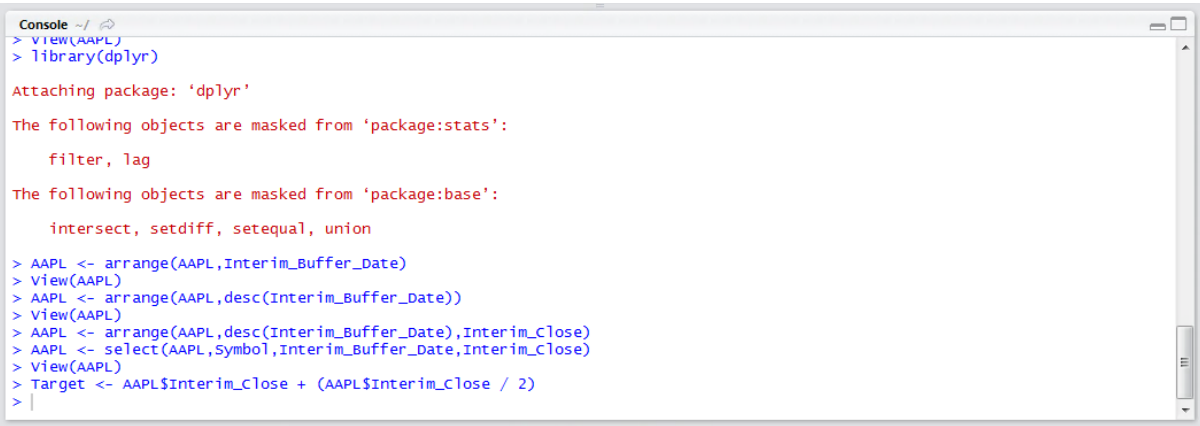
Procedure 13: Adding Vectors \ Factors to an existing Data Frame.

Abstraction is a core part of the machine learning task and horizontal abstraction would see the creation of many columns which rely on the foundational columns. In this example, a target of 50% uplift on the current price will be created as a separate column called Target (i.e. Interim_Close + (Interim_Close / 2). Firstly, create a vector which performs the formula on the Interim_Close value of the data frame AAPL by typing:

Target = AAPL$Interim_Close + (AAPL$Interim_Close / 2)

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
```

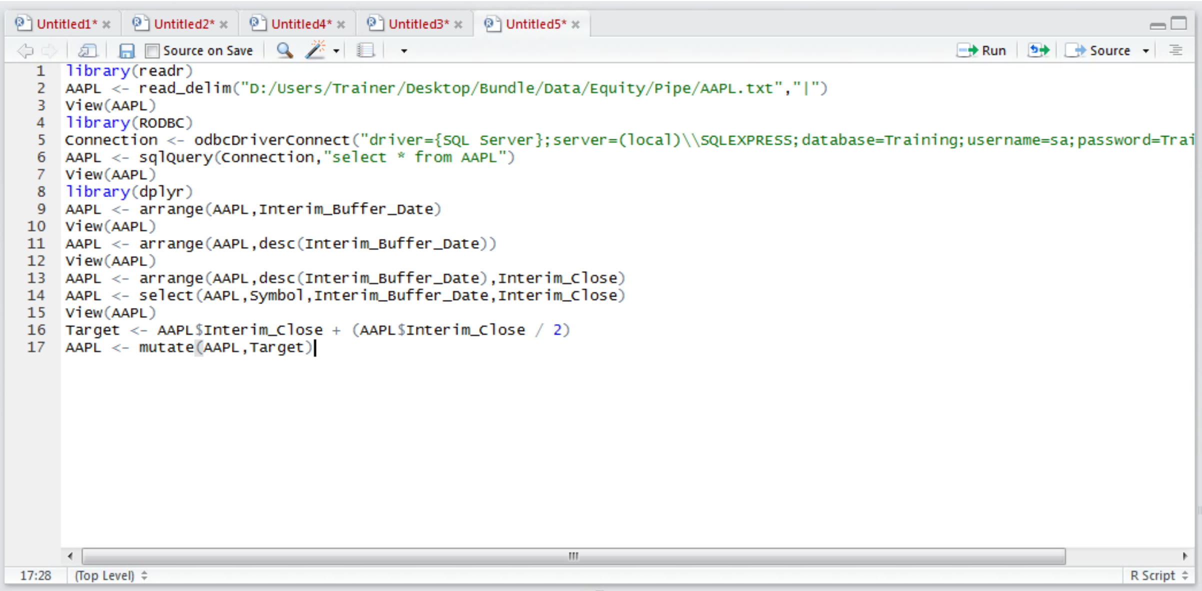
Run the line of script to console:



```
Console ~/
> View(AAPL)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
> AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
> View(AAPL)
> Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
> 
```

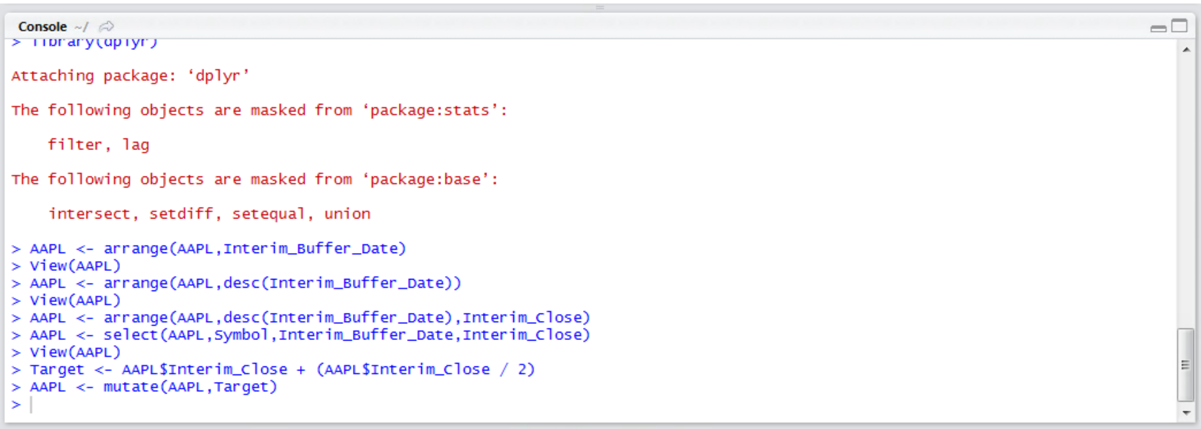
To add the column to the AAPL data frame use the mutate() function which takes the target data frame as first argument, followed by the column to added:

AAPL <- mutate(AAPL,Target)



```
Untitled1* Untitled2* Untitled4* Untitled3* Untitled5*
Source on Save   Run   Source
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
AAPL <- mutate(AAPL,Target)
17:28   (Top Level)   R Script
```

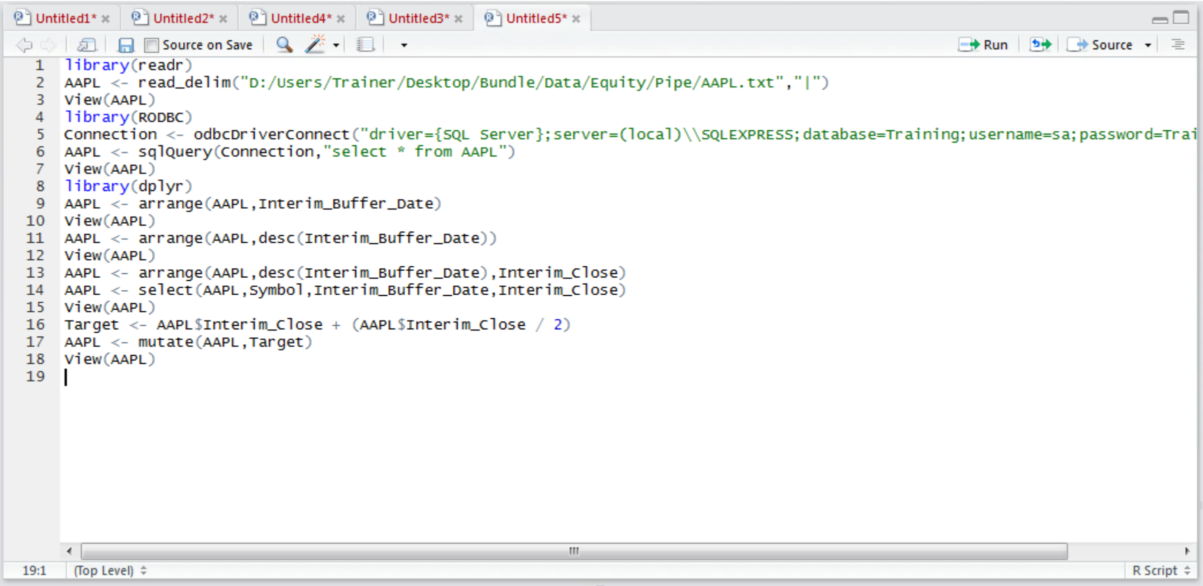
Run the line of script to console:



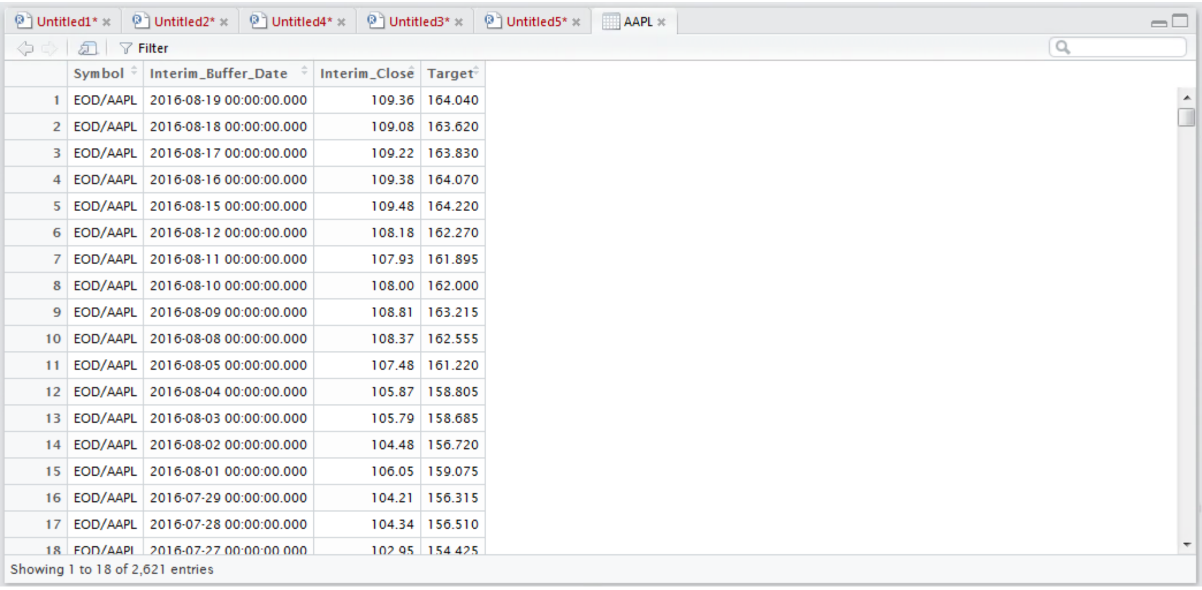
```
Console ~/
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
> AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
> View(AAPL)
> Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
> AAPL <- mutate(AAPL,Target)
> 
```

View the newly created column by typing:

View(AAPL)



Run the line of script to console to expand the data viewer in the script window:



It can be observed that the vector has been added to the data frame. The mutate() function is by far the most useful function in the creation of abstractions, whereby a vector is created via several steps, with the final vector being mutated into a Target data frame.

Procedure 14: Merging a Data Frame.

Repeat the process to create a data frame as procedure 49, this time creating a data frame called Descriptions from the table EOD_Descriptions by typing:

Descriptions <- sqlQuery(Connection,"select * from EOD_Desccriptions")

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
AAPL <- mutate(AAPL,Target)
View(AAPL)
Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
```

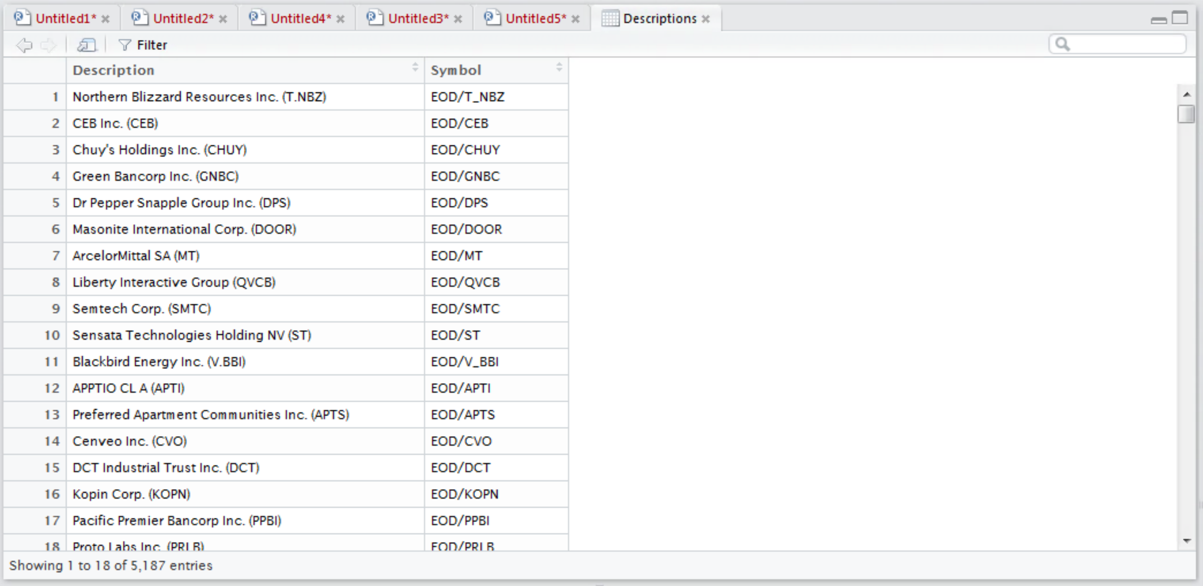
Run the line of script to console:

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
> AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
> View(AAPL)
> Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
> AAPL <- mutate(AAPL,Target)
> View(AAPL)
> Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
>
```

View the Descriptions data frame by typing:

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
AAPL <- mutate(AAPL,Target)
View(AAPL)
Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
View(Descriptions)
```

Run the line of script to console:



Untitled1* | Untitled2* | Untitled4* | Untitled3* | Untitled5* | Descriptions

Filter

	Description	Symbol
1	Northern Blizzard Resources Inc. (T.NBZ)	EOD/T_NBZ
2	CEB Inc. (CEB)	EOD/CEB
3	Chuy's Holdings Inc. (CHUY)	EOD/CHUY
4	Green Bancorp Inc. (GNBC)	EOD/GNBC
5	Dr Pepper Snapple Group Inc. (DPS)	EOD/DPS
6	Masonite International Corp. (DOOR)	EOD/DOOR
7	ArcelorMittal SA (MT)	EOD/MT
8	Liberty Interactive Group (QVCB)	EOD/QVCB
9	Semtech Corp. (SMTC)	EOD/SMTC
10	Sensata Technologies Holding NV (ST)	EOD/ST
11	Blackbird Energy Inc. (V.BBI)	EOD/V_BBI
12	APPTIO CL A (APTI)	EOD/APTI
13	Preferred Apartment Communities Inc. (APTS)	EOD/APTS
14	Cenveo Inc. (CVO)	EOD/CVO
15	DCT Industrial Trust Inc. (DCT)	EOD/DCT
16	Kopin Corp. (KOPN)	EOD/KOPN
17	Pacific Premier Bancorp Inc. (PPBI)	EOD/PPBI
18	Proto Labs Inc. (PRLB)	EOD/PRLB

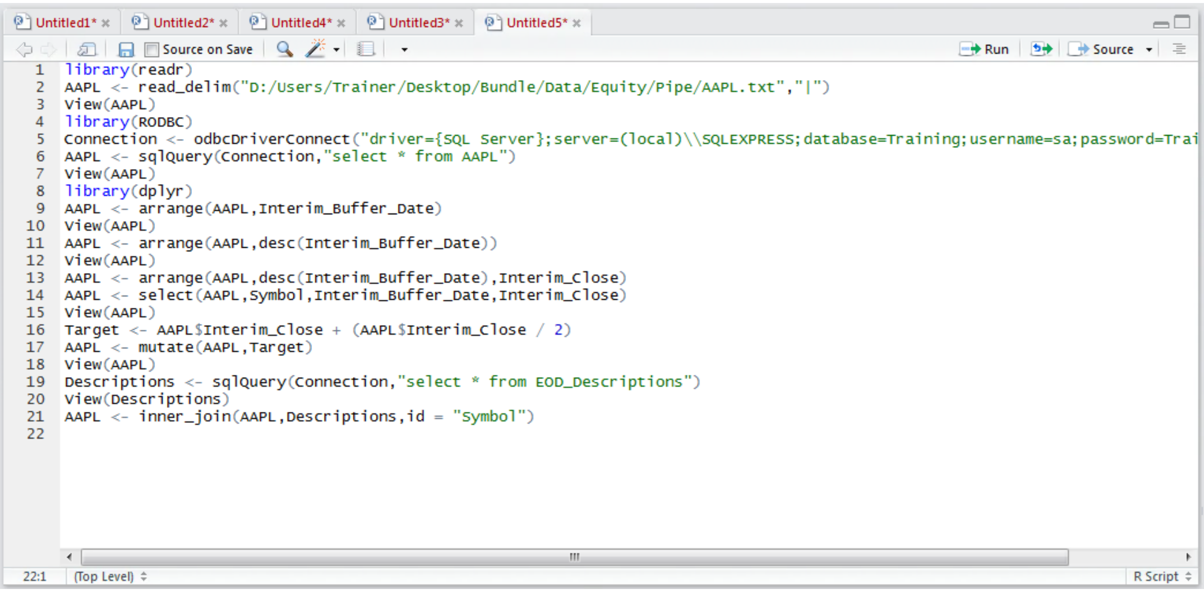
Showing 1 to 18 of 5,187 entries

It can be seen that symbol column is common between the AAPL table and the Descriptions table.

The task in this procedure is to merge the data frames together on the Symbol identifier, which will then provide a description next to each and every record in the AAPL dataset. The inner_join() function seeks to bring together all records where the key in one data frame is present in the other.

To join two data frames in this manner type:

AAPL <- inner_join(AAPL,Descriptions,ID = "Symbol")



Untitled1* | Untitled2* | Untitled4* | Untitled3* | Untitled5*

Source on Save | Run | Source

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
AAPL <- mutate(AAPL,Target)
View(AAPL)
Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
View(Descriptions)
AAPL <- inner_join(AAPL,Descriptions,id = "Symbol")

```

22:1 (Top Level) R Script

Run the line of script to console:

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> AAPL <- arrange(AAPL,Interim_Buffer_Date)
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
> View(AAPL)
> AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
> AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
> View(AAPL)
> Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
> AAPL <- mutate(AAPL,Target)
> View(AAPL)
> Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
> View(Descriptions)
> AAPL <- inner_join(AAPL,Descriptions,id = "Symbol")
Joining, by = "Symbol"
Warning message:
In inner_join_impl(x, y, by$x, by$y, suffix$x, suffix$y) :
  joining factors with different levels, coercing to character vector
> 
```

Notice that an error relating to levels has been produced, this is owing to there being a disparity in the number of records in one table as opposed to the next. Inspect the new dataset by typing:

View(AAPl)

```
library(readr)
AAPL <- read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
library(RODBC)
Connection <- odbcDriverConnect("driver={SQL Server};server=(local)\\SQLEXPRESS;database=Training;username=sa;password=Trai
AAPL <- sqlQuery(Connection,"select * from AAPL")
View(AAPL)
library(dplyr)
AAPL <- arrange(AAPL,Interim_Buffer_Date)
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date))
View(AAPL)
AAPL <- arrange(AAPL,desc(Interim_Buffer_Date),Interim_Close)
AAPL <- select(AAPL,Symbol,Interim_Buffer_Date,Interim_Close)
View(AAPL)
Target <- AAPL$Interim_Close + (AAPL$Interim_Close / 2)
AAPL <- mutate(AAPL,Target)
View(AAPL)
Descriptions <- sqlQuery(Connection,"select * from EOD_Descriptions")
View(Descriptions)
AAPL <- inner_join(AAPL,Descriptions,id = "Symbol")
View(AAPL)
```

It can be seen that the description field from the Descriptions Data Frame has been duplicated across each record in the AAPL Data Frame, as would be expected of an Inner Join in a database:

	Symbol	Interim_Buffer_Date	Interim_Close	Target	Description
1	EOD/AAPL	2016-08-19 00:00:00.000	109.36	164.040	Apple Inc. (AAPL)
2	EOD/AAPL	2016-08-18 00:00:00.000	109.08	163.620	Apple Inc. (AAPL)
3	EOD/AAPL	2016-08-17 00:00:00.000	109.22	163.830	Apple Inc. (AAPL)
4	EOD/AAPL	2016-08-16 00:00:00.000	109.38	164.070	Apple Inc. (AAPL)
5	EOD/AAPL	2016-08-15 00:00:00.000	109.48	164.220	Apple Inc. (AAPL)
6	EOD/AAPL	2016-08-12 00:00:00.000	108.18	162.270	Apple Inc. (AAPL)
7	EOD/AAPL	2016-08-11 00:00:00.000	107.93	161.895	Apple Inc. (AAPL)
8	EOD/AAPL	2016-08-10 00:00:00.000	108.00	162.000	Apple Inc. (AAPL)
9	EOD/AAPL	2016-08-09 00:00:00.000	108.81	163.215	Apple Inc. (AAPL)
10	EOD/AAPL	2016-08-08 00:00:00.000	108.37	162.555	Apple Inc. (AAPL)
11	EOD/AAPL	2016-08-05 00:00:00.000	107.48	161.220	Apple Inc. (AAPL)
12	EOD/AAPL	2016-08-04 00:00:00.000	105.87	158.805	Apple Inc. (AAPL)
13	EOD/AAPL	2016-08-03 00:00:00.000	105.79	158.685	Apple Inc. (AAPL)
14	EOD/AAPL	2016-08-02 00:00:00.000	104.48	156.720	Apple Inc. (AAPL)
15	EOD/AAPL	2016-08-01 00:00:00.000	106.05	159.075	Apple Inc. (AAPL)
16	EOD/AAPL	2016-07-29 00:00:00.000	104.21	156.315	Apple Inc. (AAPL)
17	EOD/AAPL	2016-07-28 00:00:00.000	104.34	156.510	Apple Inc. (AAPL)
18	EOD/AAPL	2016-07-27 00:00:00.000	102.95	154.425	Apple Inc. (AAPL)

Showing 1 to 18 of 2,621 entries

Procedure 15: Delete a Vector from a Data Frame.

In these procedures, the mutate() function of dplyr has been used to add a vector into a data frame. It is worthy of a brief mention that to remove a vector from a data frame, it is simply a matter of passing NULL to the vector in question:

```
AAPL$Target <- NULL
```

These procedures to not make mention to the deletion of vectors from a data frame, rather it is mentioned only for completeness.

Procedure 16: Exporting a csv file.

By this stage a large amount of manipulation has been performed on the AAPL data frame and it bears little resemblance to that which was originally loaded. Exporting data frames from R is a common requirement to communicate work product to business users. In general, if there is an object to read something into R, then there is the near equivalent to write from R. In this example, the write.csv function will be used to write the AAPL dataframe to a csv file, in the file system.

```
write.csv(AAPL,file="AAPL.csv")
```

```r
DoubleNested <- sqrt(round(Double,2))
Names <- c("Richard","Robert","Reinhard","Raymond","Richardino","Richardo")
NamesSubstr <= substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl(^Richard,NamesSubstr)
NamesGrepl
NameGreplNumeric <- as.numeric(NamesGrepl)
NamesGrepNumeric
NamesGrepNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
Dates <- as.Date(DatesString,format="%d/%m/%Y")
Dates
DatesPlusOne <- Dates + 1
DatesPlusOne
ReportingQuarters <- quarters(Dates)
ReportingQuarters
AAPL <- Read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
write.csv(AAPL,file="AAPL.csv")
```

Run the line of script to console:

```
> library(readr)
> AAPL <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Training/NoNetica/Data/Equi
ty/Equity/AAPL.csv")
Parsed with column specification:
cols(
  Symbol = col_character(),
  Interim_Buffer_Date = col_datetime(format = ""),
  Interim_Open = col_double(),
  Interim_Low = col_double(),
  Interim_High = col_double(),
  Interim_Close = col_double()
)
> View(AAPL)
> write.csv(AAPL,file="AAPL.csv")
>
```

To identify the location of the working directory, use the getwd() function:

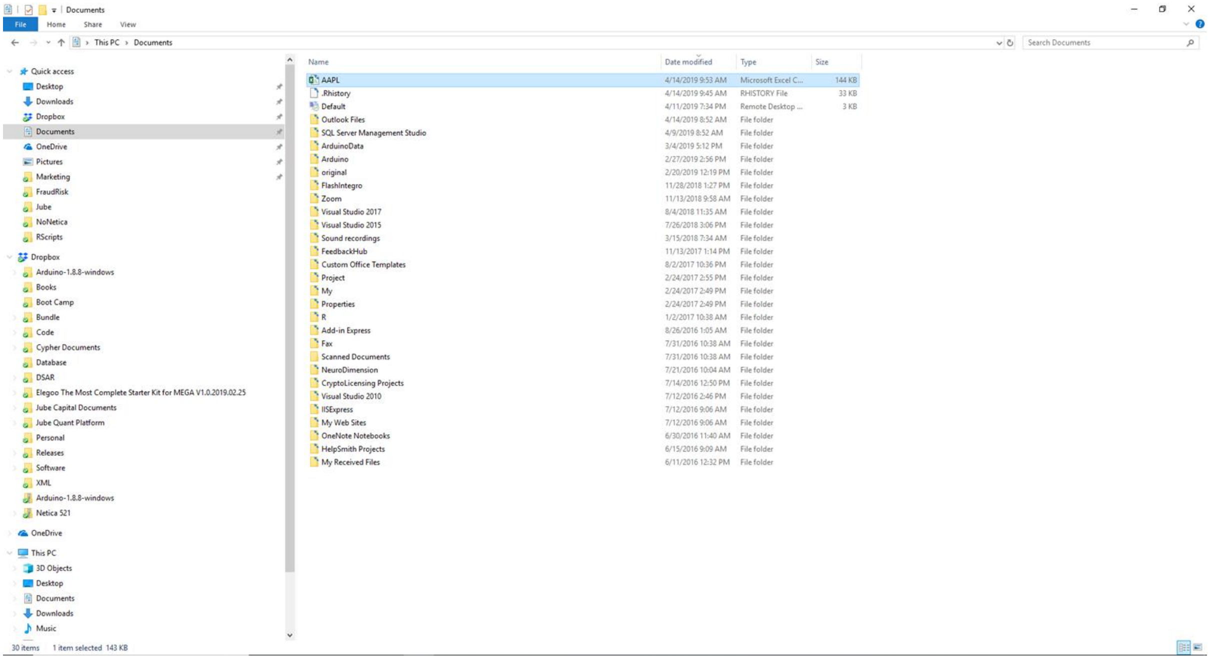
getwd()

```
Names <- c("Richard","Robert","Reinhard","Raymond","Richardinio","Richardo")
NamesSubstr <= substr(Names,1,7)
NamesSubstr
NamesSubstrLogical <- NamesSubstr == "Richard"
NamesSubstrLogical
NamesSubstrLogicalNumeric <- as.numeric(NamesSubstrLogical)
NamesSubstrLogicalNumericNested <- as.numeric(substr(Names,1,7) == "Richard")
NamesGrepl <- grepl(^Richard,NamesSubstr)
NamesGrepl
NameGreplNumeric <- as.numeric(NamesGrepl)
NamesGrepNumeric
NamesGrepNumericNested <- as.numeric(grepl("^Richard",Names))
DatesString <- c("22/02/1732","30/10/1735","13/04/1743","16/03/1751")
Dates <- as.Date(DatesString,format="%d/%m/%Y")
Dates
DatesPlusOne <- Dates + 1
DatesPlusOne
ReportingQuarters <- quarters(Dates)
ReportingQuarters
AAPL <- Read_delim("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Pipe/AAPL.txt","|")
View(AAPL)
write.csv(AAPL,file="AAPL.csv")
getwd()
```

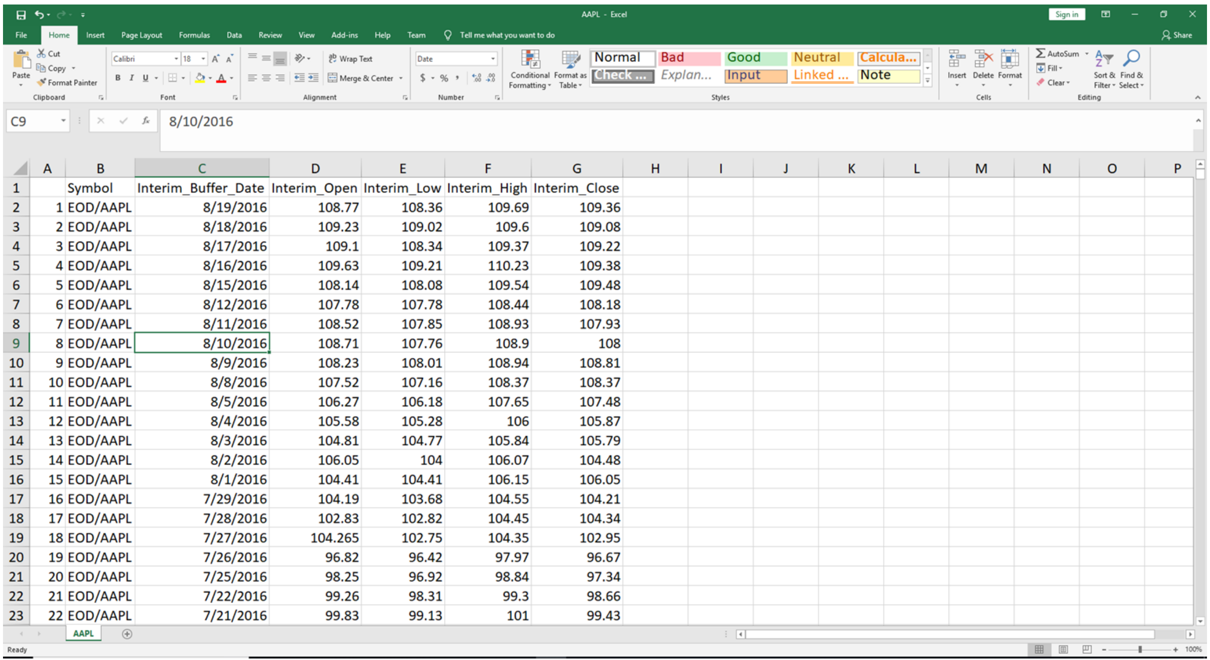
Run the line of script to console:

```
> AAPL <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Training/NoNetica/Data/Equi
ty/Equity/AAPL.csv")
Parsed with column specification:
cols(
  Symbol = col_character(),
  Interim_Buffer_Date = col_datetime(format = ""),
  Interim_Open = col_double(),
  Interim_Low = col_double(),
  Interim_High = col_double(),
  Interim_Close = col_double()
)
> View(AAPL)
> write.csv(AAPL,file="AAPL.csv")
> getwd()
[1] "C:/Users/Richard/Documents"
>
```

Open the directory in windows explorer:



Opening the file, it can be seen that the data frame has been reliably exported:

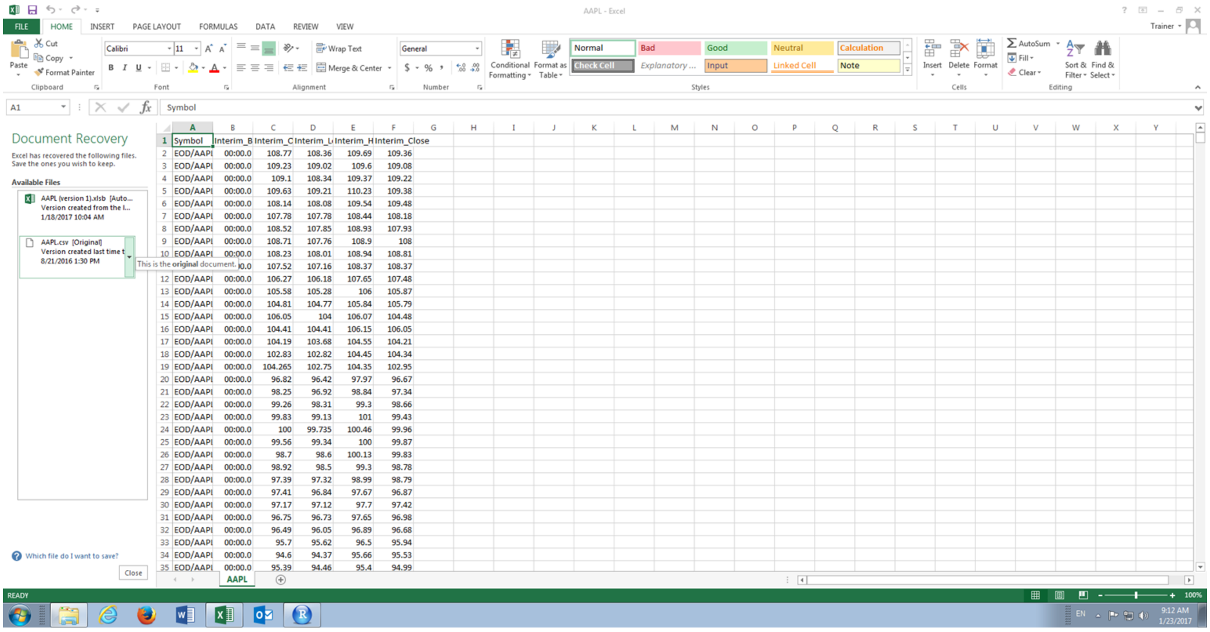


	Symbol	Interim_Buffer_Date	Interim_Open	Interim_Low	Interim_High	Interim_Close
1	EOD/AAPL	8/19/2016	108.77	108.36	109.69	109.36
2	EOD/AAPL	8/18/2016	109.23	109.02	109.6	109.08
3	EOD/AAPL	8/17/2016	109.1	108.34	109.37	109.22
4	EOD/AAPL	8/16/2016	109.63	109.21	110.23	109.38
5	EOD/AAPL	8/15/2016	108.14	108.08	109.54	109.48
6	EOD/AAPL	8/12/2016	107.78	107.78	108.44	108.18
7	EOD/AAPL	8/11/2016	108.52	107.85	108.93	107.93
8	EOD/AAPL	8/10/2016	108.71	107.76	108.9	108
9	EOD/AAPL	8/9/2016	108.23	108.01	108.94	108.81
10	EOD/AAPL	8/8/2016	107.52	107.16	108.37	108.37
11	EOD/AAPL	8/5/2016	106.27	106.18	107.65	107.48
12	EOD/AAPL	8/4/2016	105.58	105.28	106	105.87
13	EOD/AAPL	8/3/2016	104.81	104.77	105.84	105.79
14	EOD/AAPL	8/2/2016	106.05	104	106.07	104.48
15	EOD/AAPL	8/1/2016	104.41	104.41	106.15	106.05
16	EOD/AAPL	7/29/2016	104.19	103.68	104.55	104.21
17	EOD/AAPL	7/28/2016	102.83	102.82	104.45	104.34
18	EOD/AAPL	7/27/2016	104.265	102.75	104.35	102.95
19	EOD/AAPL	7/26/2016	96.82	96.42	97.97	96.67
20	EOD/AAPL	7/25/2016	98.25	96.92	98.84	97.34
21	EOD/AAPL	7/22/2016	99.26	98.31	99.3	98.66
22	EOD/AAPL	7/21/2016	99.83	99.13	101	99.43

Module 5 Summary Statistics and Basic Plots in R.

Summary statistics refer to the creation of commonly used aggregate statistics from a data frame, in this case a data frame of AAPL prices for the last ten years. In this module R will be used to load the AAPL prices then explore this data using summary statistics and some rudimentary plots.

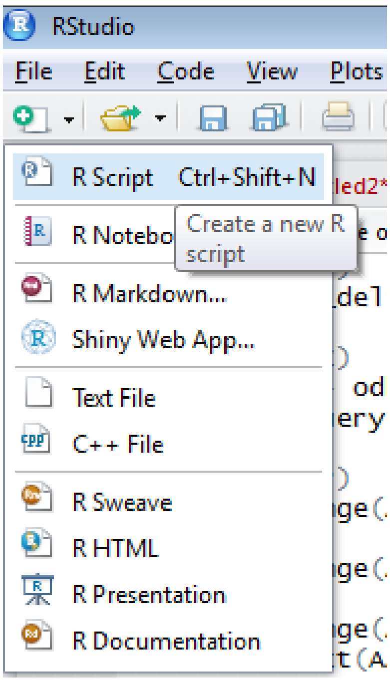
The data file to be used is the AAPL.csv file located in Bundle\Data\Equity\Equity\AAPL.csv:



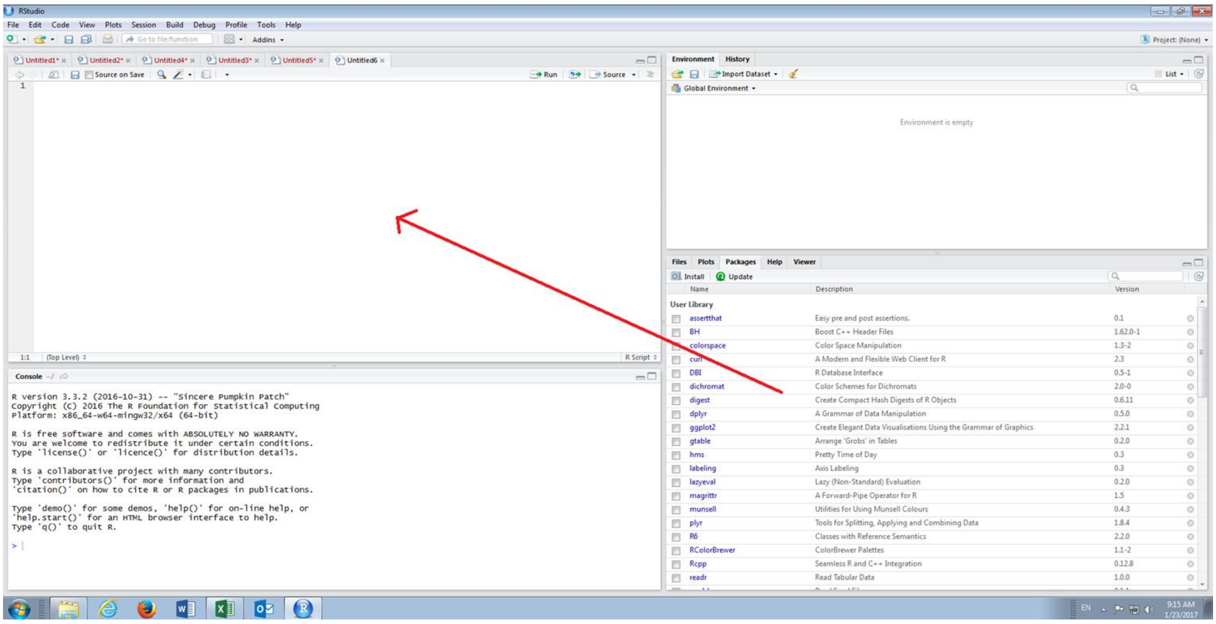
The module seeks to emulate many of the functions available to Excel and StatTools in R.

Procedure 1: Create a Histogram of Time Series Data in R.

Start this procedure by creating a new script window in RStudio by clicking on in the top left hand corner, then clicking RScript on the submenu:



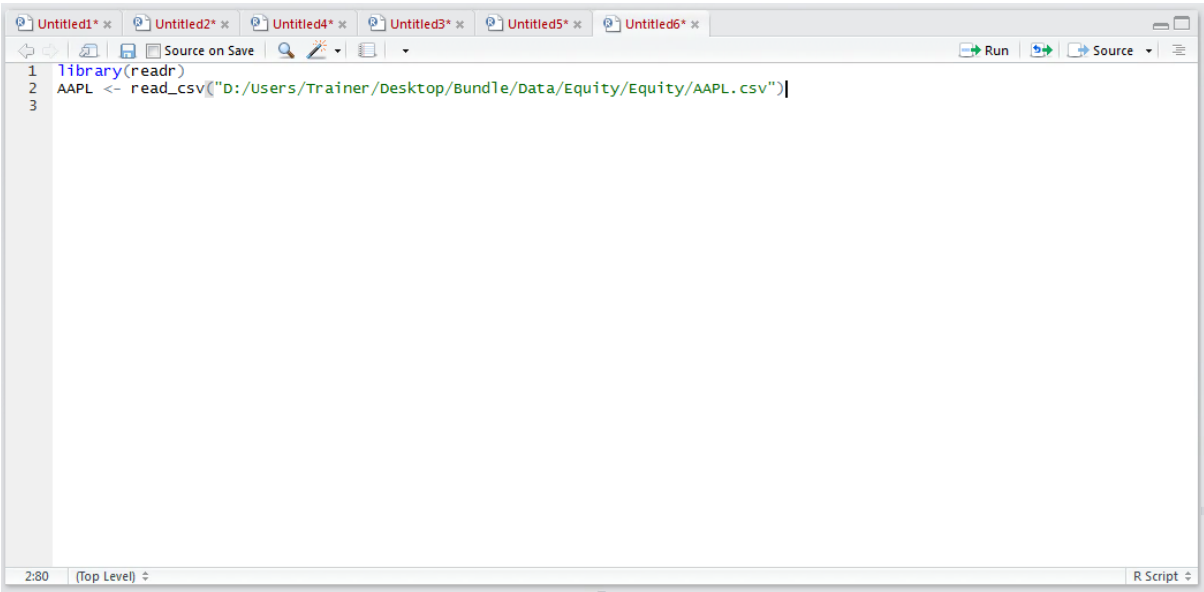
A new script window will be opened and be ready for input:



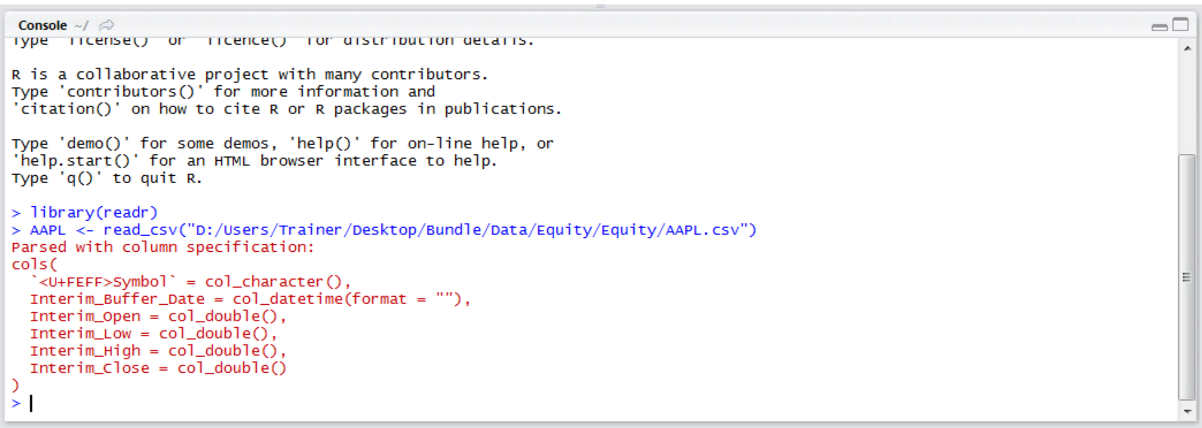
Load the AAPL.csv dataset from Bundle\Data\Equity\Equity\AAPL.csv:

library(readr)

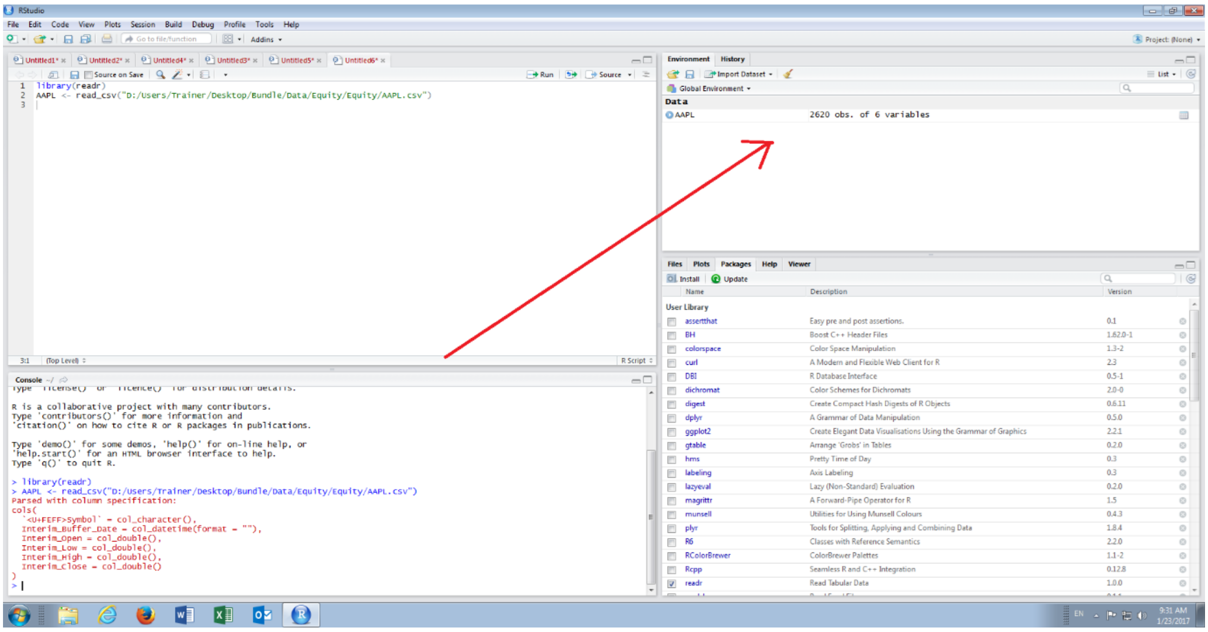
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")



It can be observed that the library readr is being loaded and therafter the read_csv() function is being used to create a data frame titled AAPL. Run the block of script to console:

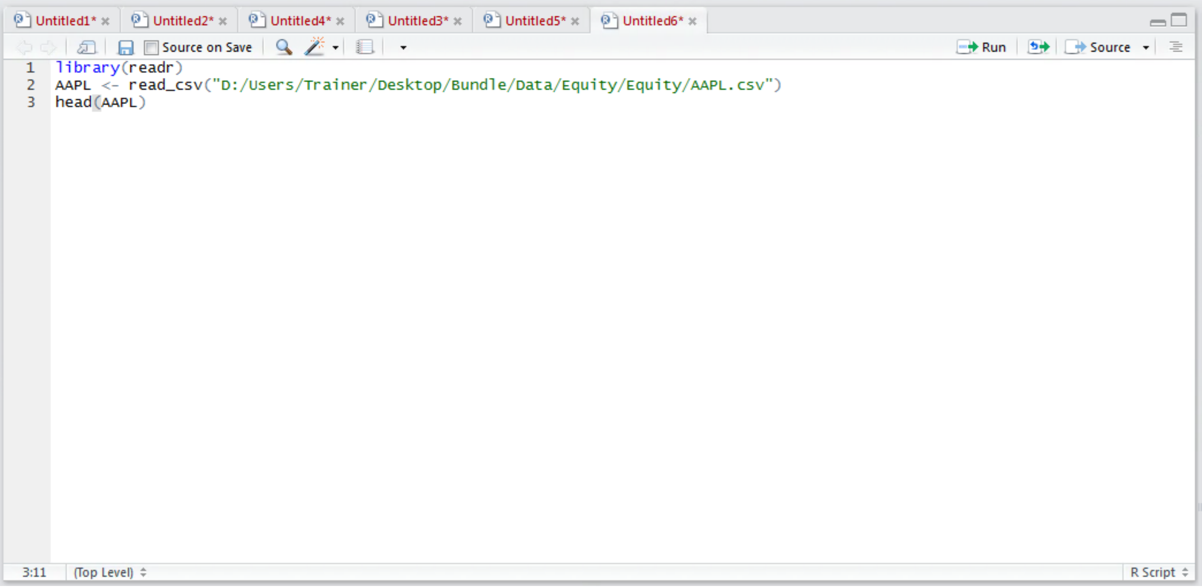


The specification has been written to console and is available in the environment pane:

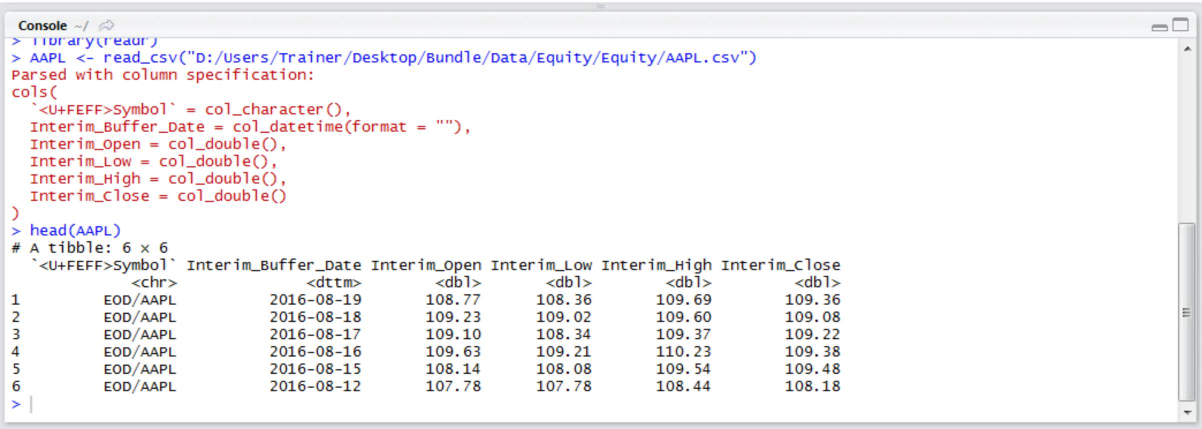


As the data frame is quite large, it is not practical to write it all out to console, hence in this example the head() function will be used to take a peek at the data frame by typing:

head(AAPL)



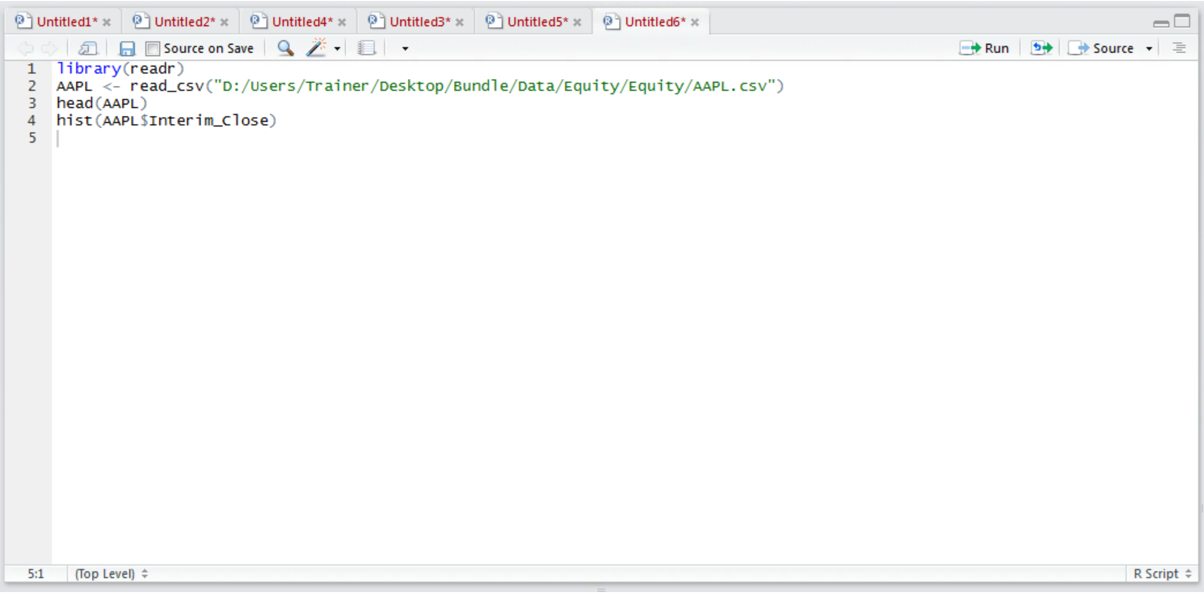
Run the line of script to console:



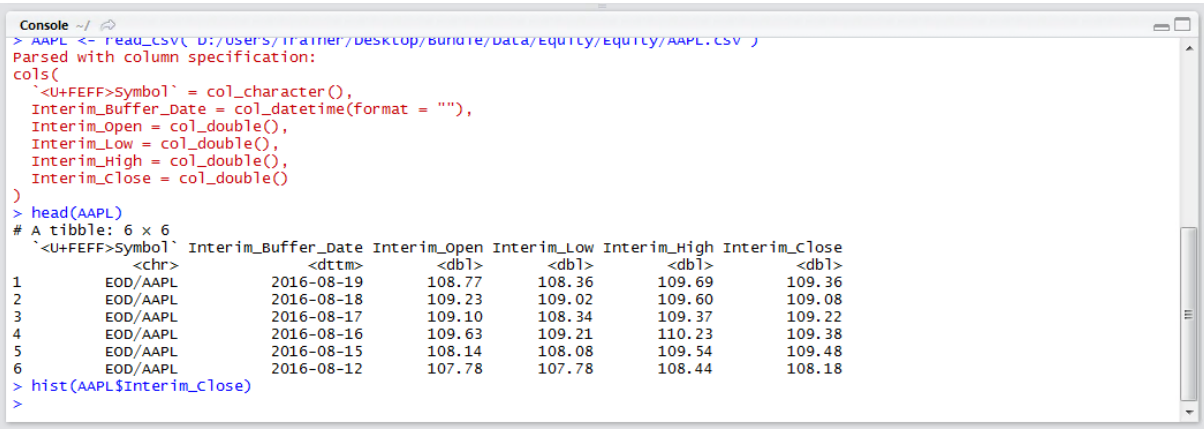
It can be seen that just the top of the data frame has been returned.

For the purposes of this procedure, the column, rather vector, of interest is the Interim_Close for which a histogram would provide some discovery capability. To create a histogram the hist() function is used, taking the data frame and named vector:

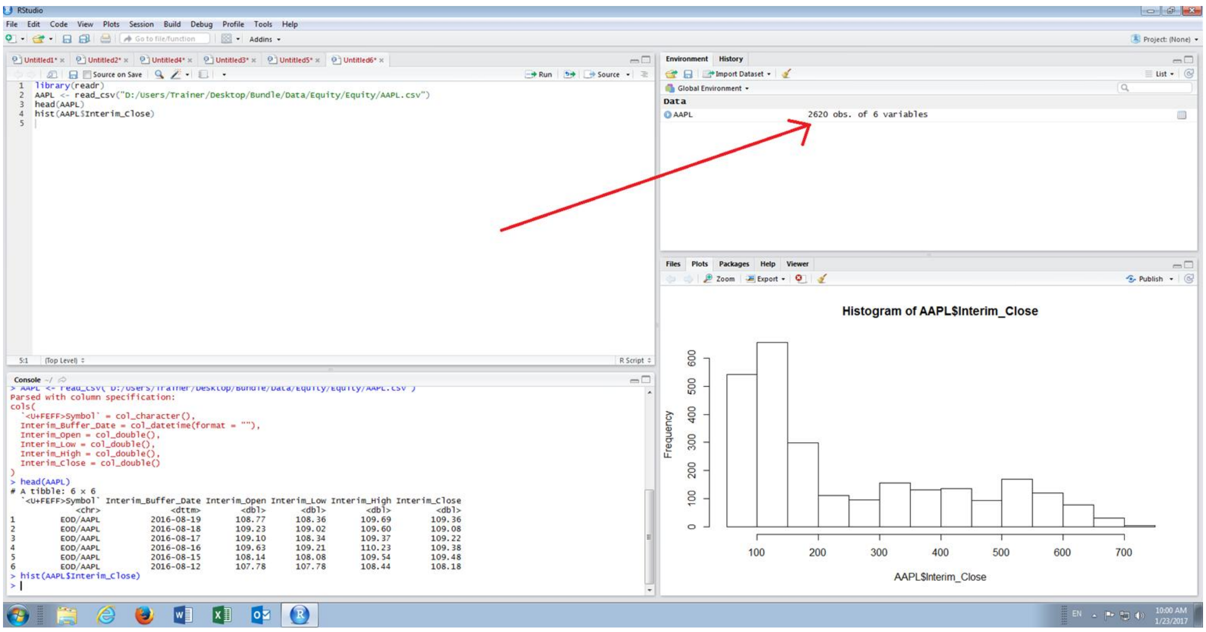
hist(AAPL$Interim_Close)



Run the line of script to console:



It can be seen that a chart has been loaded to the plot section of RStudio:

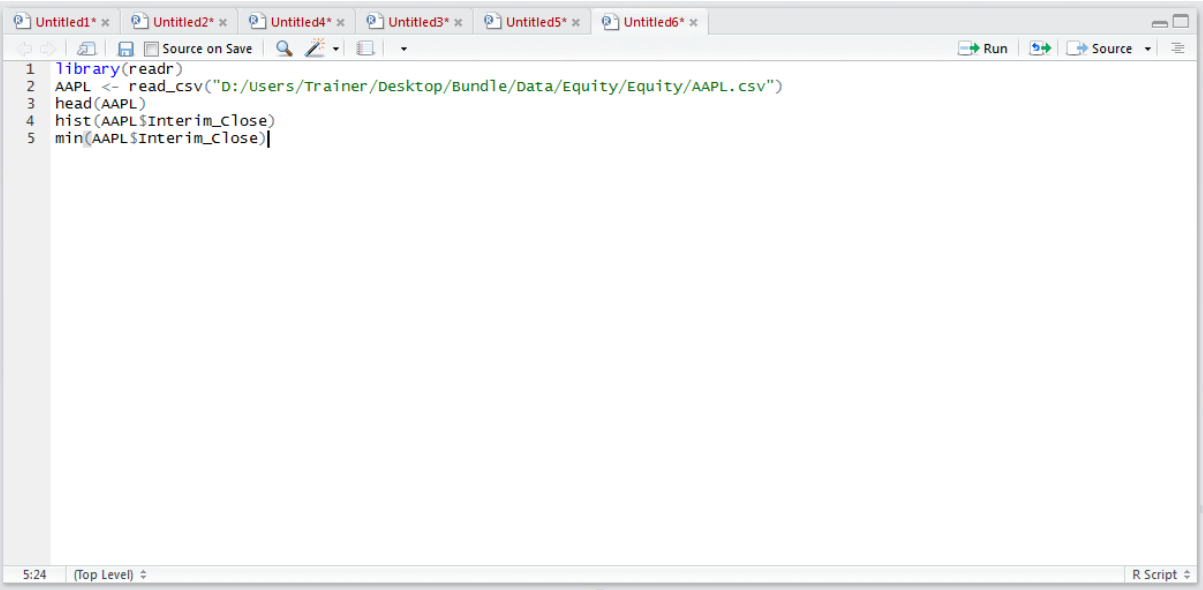


The plot gives a good snap visulatisation of the AAPL stock price over the history, which in this case can be seen as positivily skewed. The hist() function exposes many argument to enhance the visual appearance of the histogram however for the purposes of exploration, rather than presentation, the defaults are more than adequete.

Procedure 2: Establish Range in R.

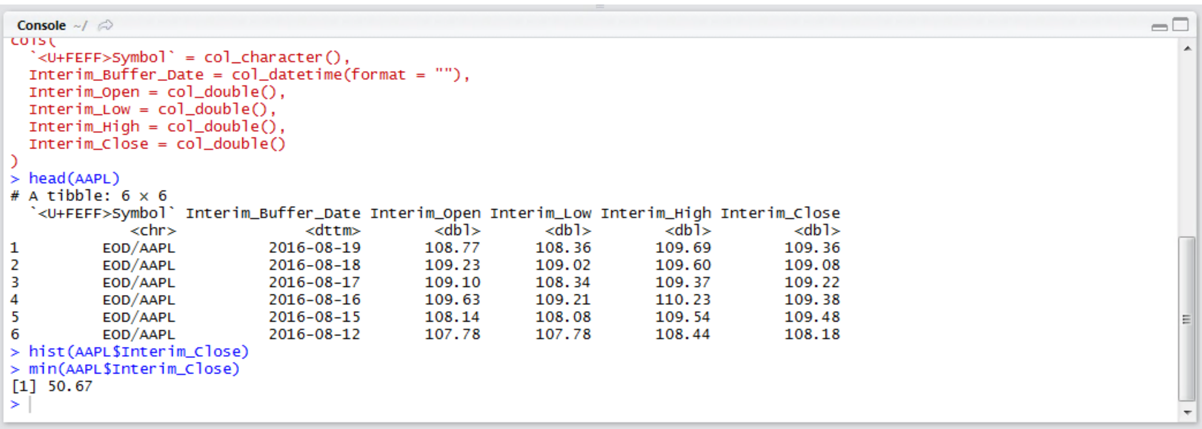
To establish the range of the Interim_Close in the AAPL data frame use the min() function typing:

min(AAPL$Interim_Close)



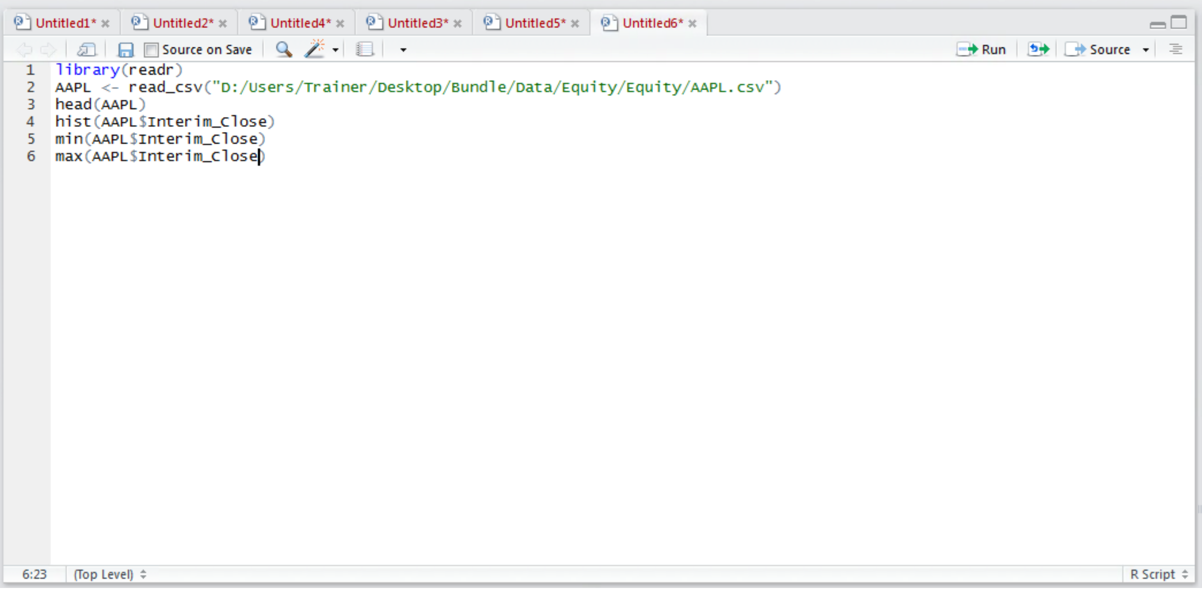
```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
```

Run the line of script to console:

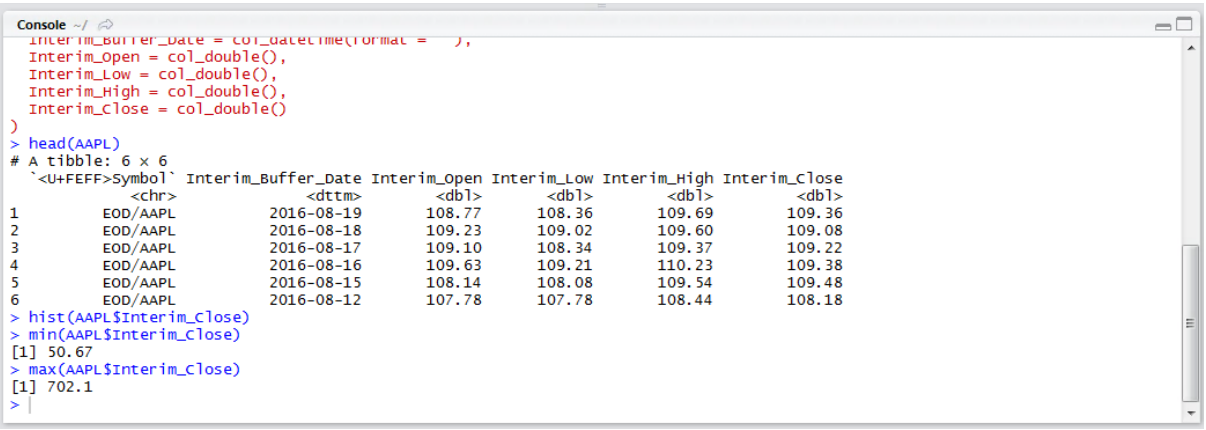


```
cols(
  `<U+FEFF>Symbol` = col_character(),
  Interim_Buffer_Date = col_datetime(format = ""),
  Interim_Open = col_double(),
  Interim_Low = col_double(),
  Interim_High = col_double(),
  Interim_Close = col_double()
)
> head(AAPL)
# A tibble: 6 x 6
  `<U+FEFF>Symbol` Interim_Buffer_Date Interim_Open Interim_Low Interim_High Interim_Close
             <chr>              <dttm>        <dbl>       <dbl>        <dbl>         <dbl>
1         EOD/AAPL          2016-08-19       108.77      108.36       109.69        109.36
2         EOD/AAPL          2016-08-18       109.23      109.02       109.60        109.08
3         EOD/AAPL          2016-08-17       109.10      108.34       109.37        109.22
4         EOD/AAPL          2016-08-16       109.63      109.21       110.23        109.38
5         EOD/AAPL          2016-08-15       108.14      108.08       109.54        109.48
6         EOD/AAPL          2016-08-12       107.78      107.78       108.44        108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
>
```

It can be seen that the smallest value in the Interim_Close vector of the AAPL data frame is 50.67, to retrieve the largest value use the max() function by typing:

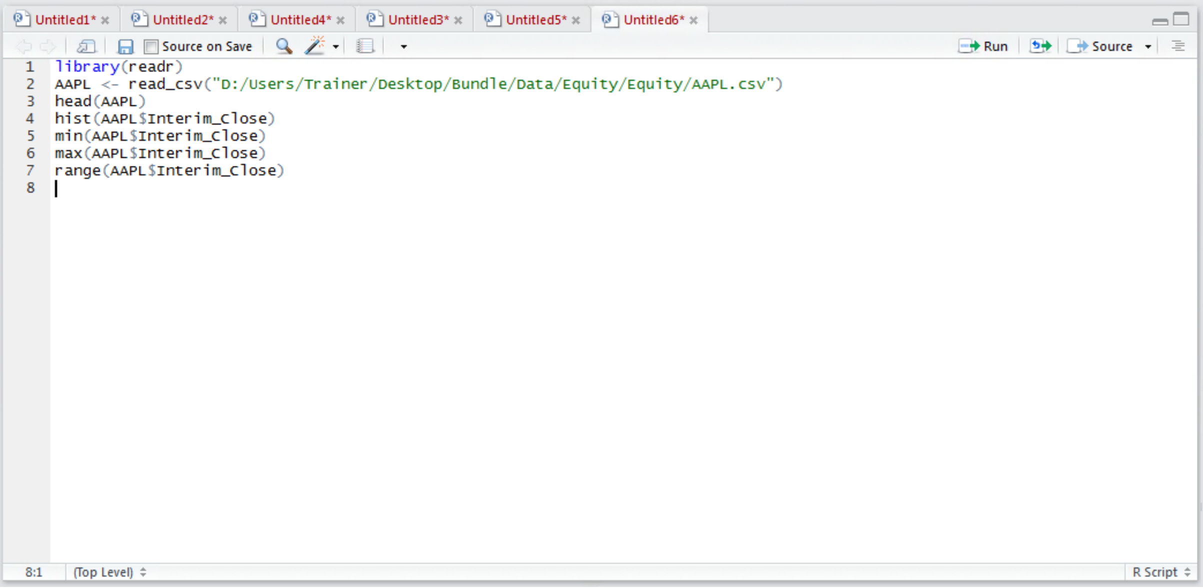


Run the line of script to console:



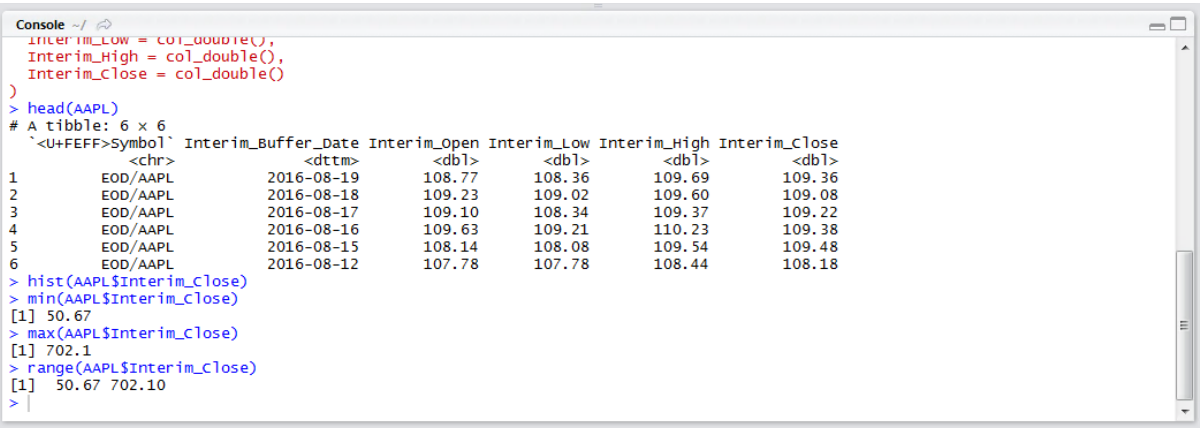
It can be observed from the console that the largest price is 702.1. The range can be calculated by subtracting the maximum value from the minimum value. The values can be presented more succinctly using the range() function and typing:

range(AAPL$Interim_Close)



```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
```

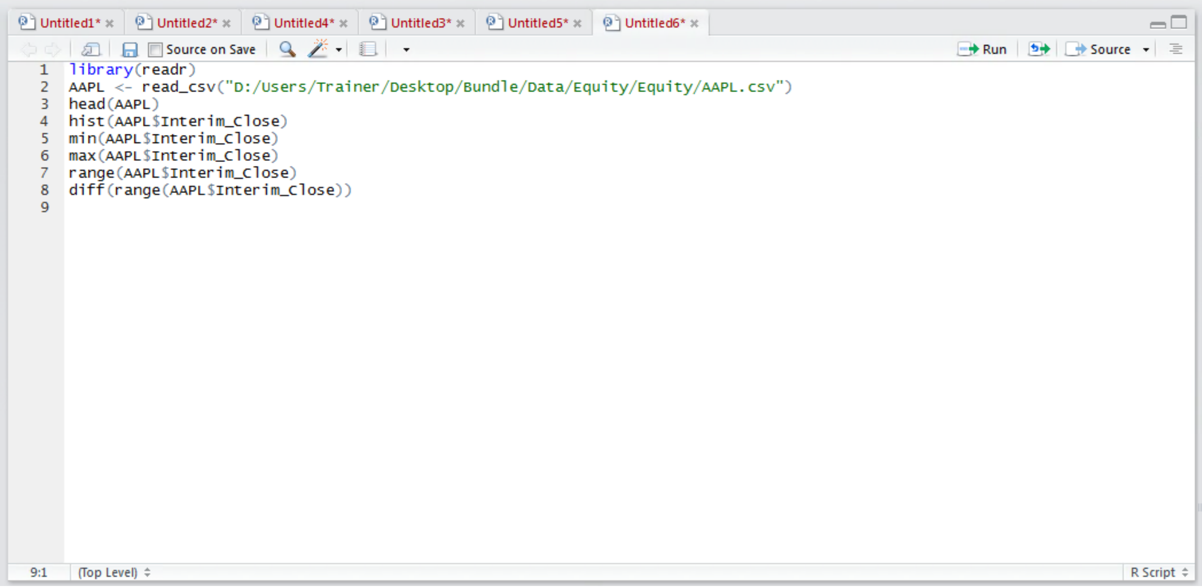
Run the line of script to console:



```
  Interim_Low = col_double(),
  Interim_High = col_double(),
  Interim_Close = col_double()
)
> head(AAPL)
# A tibble: 6 x 6
  `<U+FEFF>Symbol` Interim_Buffer_Date Interim_Open Interim_Low Interim_High Interim_Close
             <chr>              <dttm>        <dbl>       <dbl>        <dbl>         <dbl>
1         EOD/AAPL          2016-08-19       108.77      108.36       109.69        109.36
2         EOD/AAPL          2016-08-18       109.23      109.02       109.60        109.08
3         EOD/AAPL          2016-08-17       109.10      108.34       109.37        109.22
4         EOD/AAPL          2016-08-16       109.63      109.21       110.23        109.38
5         EOD/AAPL          2016-08-15       108.14      108.08       109.54        109.48
6         EOD/AAPL          2016-08-12       107.78      107.78       108.44        108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
>
```

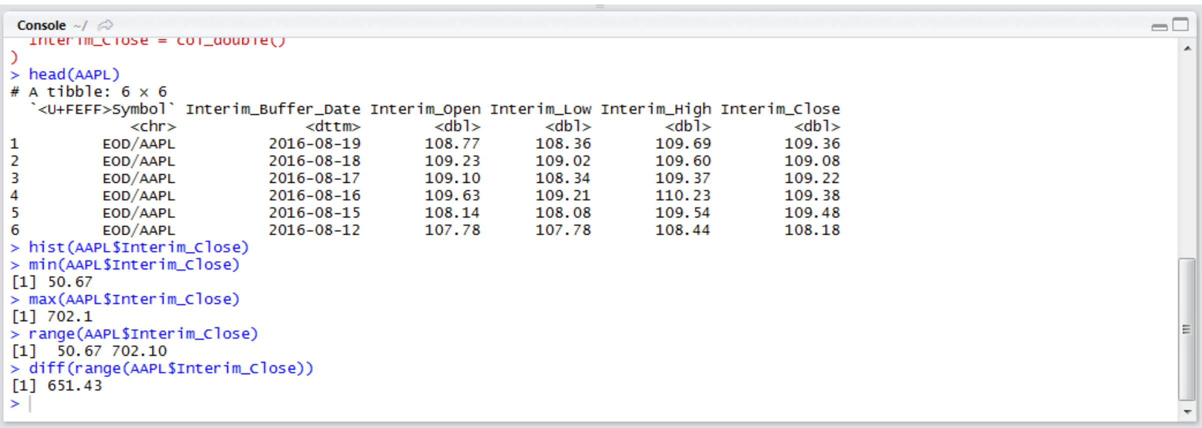
To establish the range value subtract the largest value from the smallest value which can be achived by using the diff() function on the vector returned from the range() function as:

diff(range(AAPL$Interval_Close))



```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
```

Run the line of script to console:



```
  Interim_Close = col_double()
)
> head(AAPL)
# A tibble: 6 x 6
  `<U+FEFF>Symbol` Interim_Buffer_Date Interim_Open Interim_Low Interim_High Interim_Close
             <chr>              <dttm>        <dbl>       <dbl>        <dbl>         <dbl>
1         EOD/AAPL          2016-08-19       108.77      108.36       109.69        109.36
2         EOD/AAPL          2016-08-18       109.23      109.02       109.60        109.08
3         EOD/AAPL          2016-08-17       109.10      108.34       109.37        109.22
4         EOD/AAPL          2016-08-16       109.63      109.21       110.23        109.38
5         EOD/AAPL          2016-08-15       108.14      108.08       109.54        109.48
6         EOD/AAPL          2016-08-12       107.78      107.78       108.44        108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
>
```

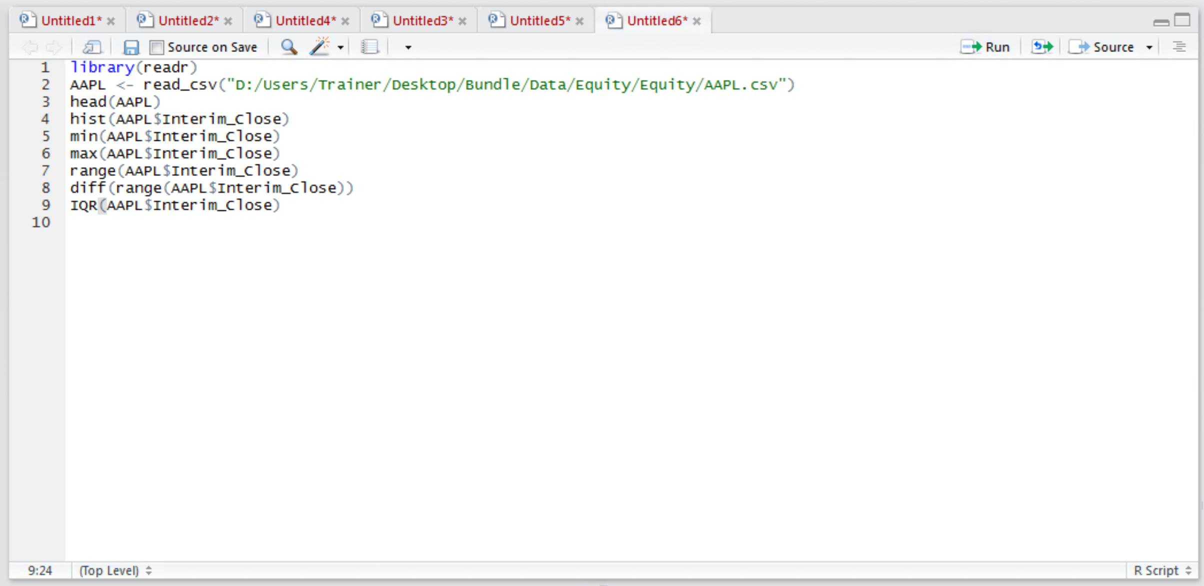
It can be seen that the range has been returned as being 651.43.

Procedure 3: Calculate Quartiles and the Interquartile Range.

Quartiles, which divides the vector up into four chunks which are equally sized, is one means to estimate spread. The IQR() function allocates the entries in a vector and provides explanation of the thresholds, returning the range between the end of the first quartile and the start of the third quartile. To establish quartiles type:

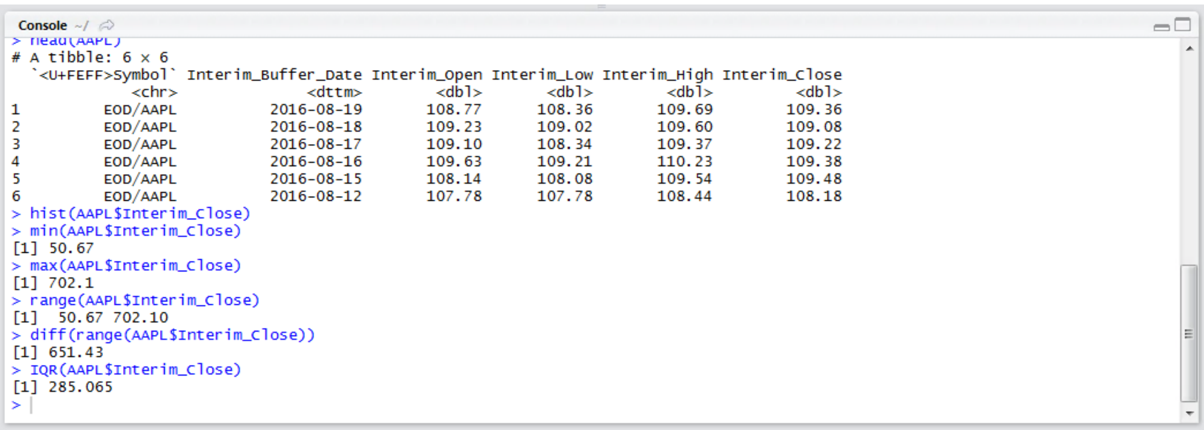
IQR(AAPL$Interim_Close)

JUBE



```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
```

Run the line of script to console:



```
> head(AAPL)
# A tibble: 6 x 6
  `<U+FEFF>Symbol` Interim_Buffer_Date Interim_Open Interim_Low Interim_High Interim_Close
             <chr>              <dttm>        <dbl>       <dbl>        <dbl>         <dbl>
1         EOD/AAPL          2016-08-19       108.77      108.36       109.69        109.36
2         EOD/AAPL          2016-08-18       109.23      109.02       109.60        109.08
3         EOD/AAPL          2016-08-17       109.10      108.34       109.37        109.22
4         EOD/AAPL          2016-08-16       109.63      109.21       110.23        109.38
5         EOD/AAPL          2016-08-15       108.14      108.08       109.54        109.48
6         EOD/AAPL          2016-08-12       107.78      107.78       108.44        108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
>
```

To obtain more granularity around the range calculated using the IQR() function, use the quantile() function by typing:

quantile(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
```

Run the line of script to console:

```
          <chr>           <date>     <dbl>    <dbl>    <dbl>     <dbl>
1      EOD/AAPL       2016-08-19    108.77   108.36   109.69    109.36
2      EOD/AAPL       2016-08-18    109.23   109.02   109.60    109.08
3      EOD/AAPL       2016-08-17    109.10   108.34   109.37    109.22
4      EOD/AAPL       2016-08-16    109.63   109.21   110.23    109.38
5      EOD/AAPL       2016-08-15    108.14   108.08   109.54    109.48
6      EOD/AAPL       2016-08-12    107.78   107.78   108.44    108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
>
```

The first quartile is 107.510, the second quartile is 171.195 and the third quartile is 392.575, values which provide a measure of spread and coupled with other summary statistics can support a further visualization in the form of a box plot, as explained in procedure 59.

Procedure 4: Establish the Mean and Median in R.

The Mean and Median are a way to measure the central tendency of a vector. The mean, commonly called the average, is calculated by summing up all of the values in a vector the dividing it by the count of values in the vector (i.e. 100 + 200 + 300 / 3). The function mean() performs the calculation on a vector by typing:

mean(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
```

Run the line of script to console:

```
2        EOD/AAPL        2016-08-18     109.23      109.02      109.60      109.08
3        EOD/AAPL        2016-08-17     109.10      108.34      109.37      109.22
4        EOD/AAPL        2016-08-16     109.63      109.21      110.23      109.38
5        EOD/AAPL        2016-08-15     108.14      108.08      109.54      109.48
6        EOD/AAPL        2016-08-12     107.78      107.78      108.44      108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
> mean(AAPL$Interim_Close)
[1] 251.8668
>
```

The mean, or average, is output as 251.8668. The median on the other hand is absolute middle of a histogram and can be calculated using the median() function:

median(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
```

Run the line of script to console:

```
4         EOD/AAPL          2016-08-16       109.63       109.21       110.23        109.38
5         EOD/AAPL          2016-08-15       108.14       108.08       109.54        109.48
6         EOD/AAPL          2016-08-12       107.78       107.78       108.44        108.18
> hist(AAPL$Interim_Close)
> min(AAPL$Interim_Close)
[1] 50.67
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100% 
 50.670 107.510 171.195 392.575 702.100 
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> 
```

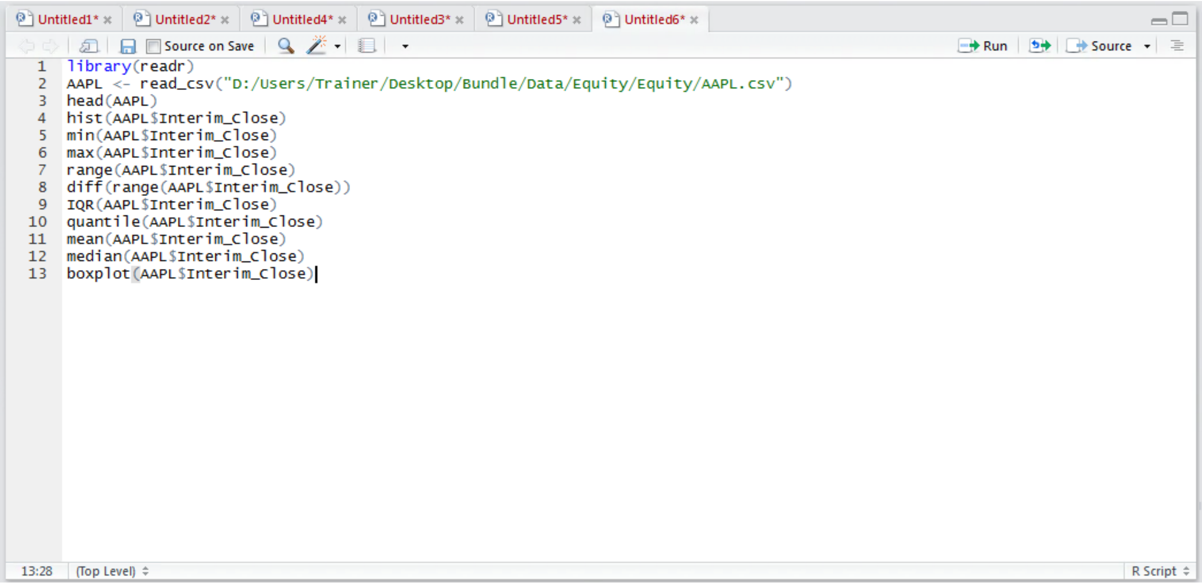
It can be observed that the median, the value that could be considered the centre of the distribution, is 171.195. Taken together with procedure 57, all the values are present to create a box and whiskers plot as an alternative to a histogram as a means to understand the spread of data, and is explained in procedure 59.

Procedure 5: Create a Box Plot.

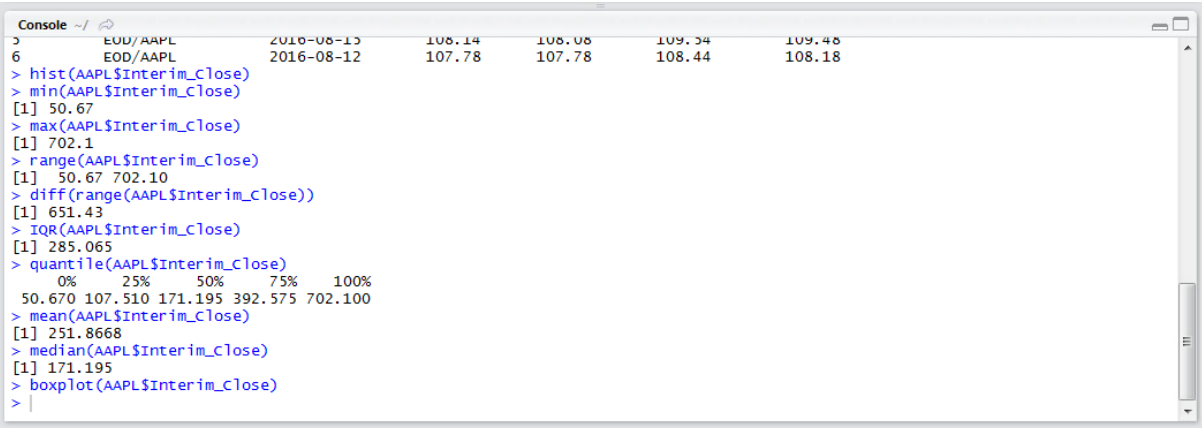
A box plot is a five-point visualisation of several summary statistics, the Median, the Range and the Quartile Range. The box plot allows for a quick appraisal of range and skew of the data and is an alternative to a histogram relying solely on easily reproducible summary statistics.

The boxplot() function takes a vector as its argument and produces a visualisation. To create a Box Plot simply type:

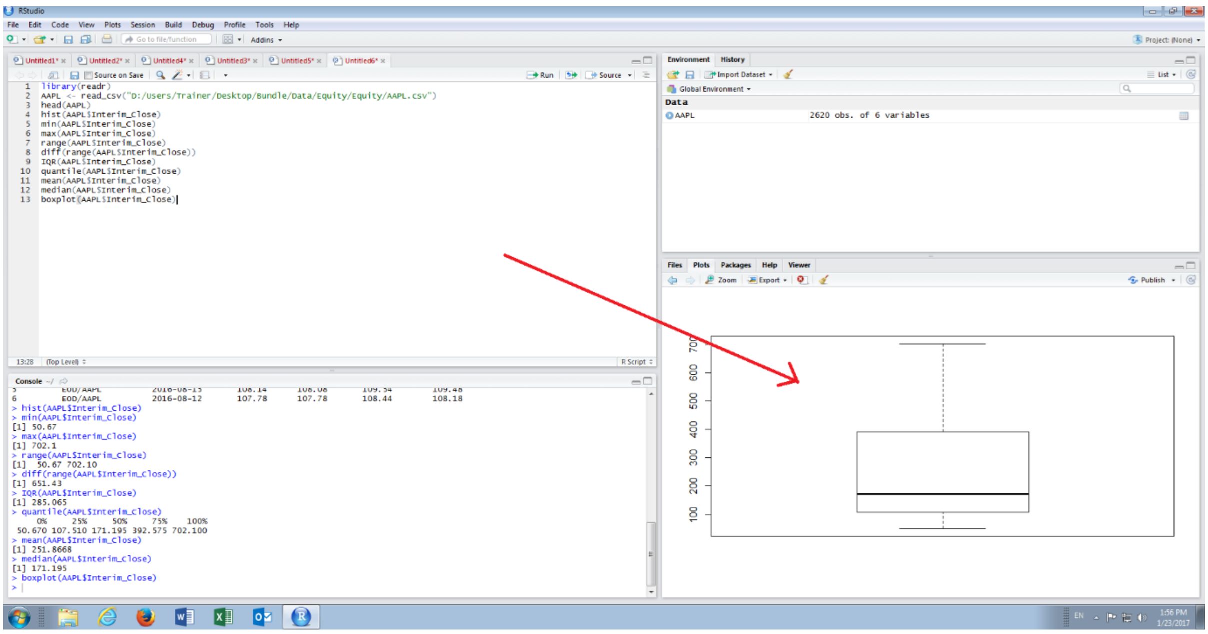
boxplot(AAPL$Interim_Close)



Run the line of script to console:



The box plot is drawn in the plots window in RStudio:

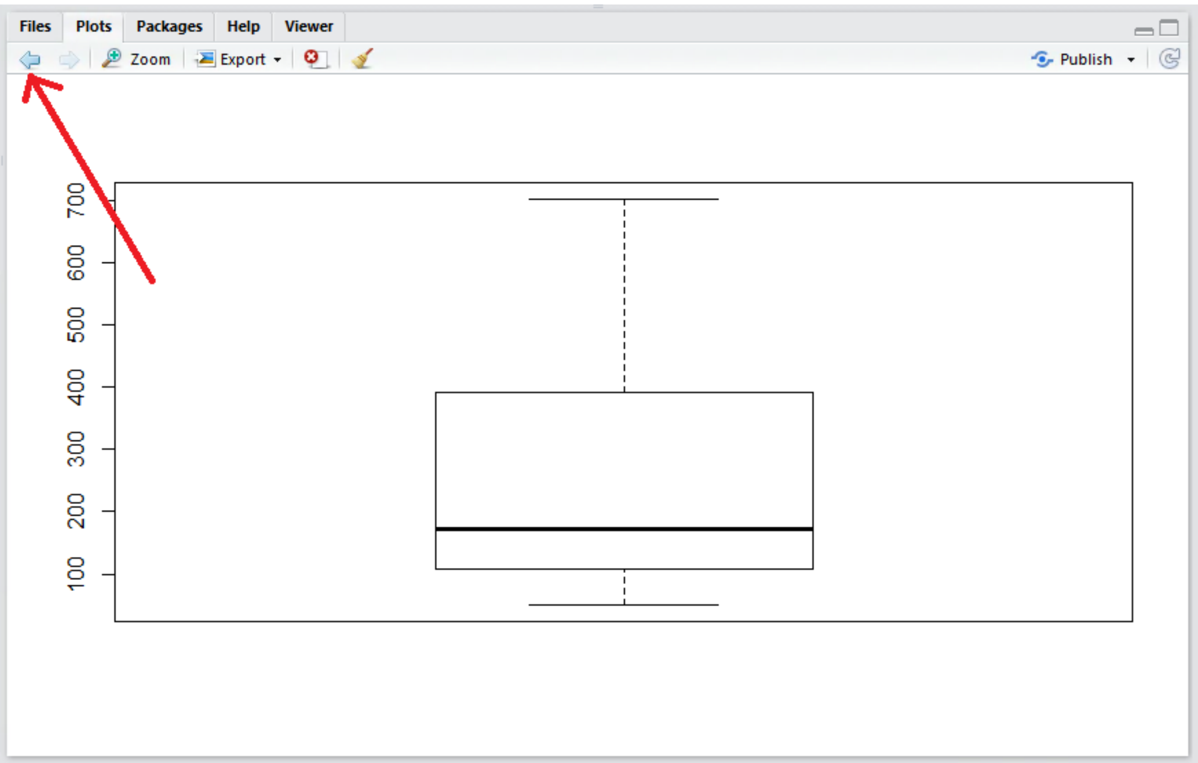


JUBE

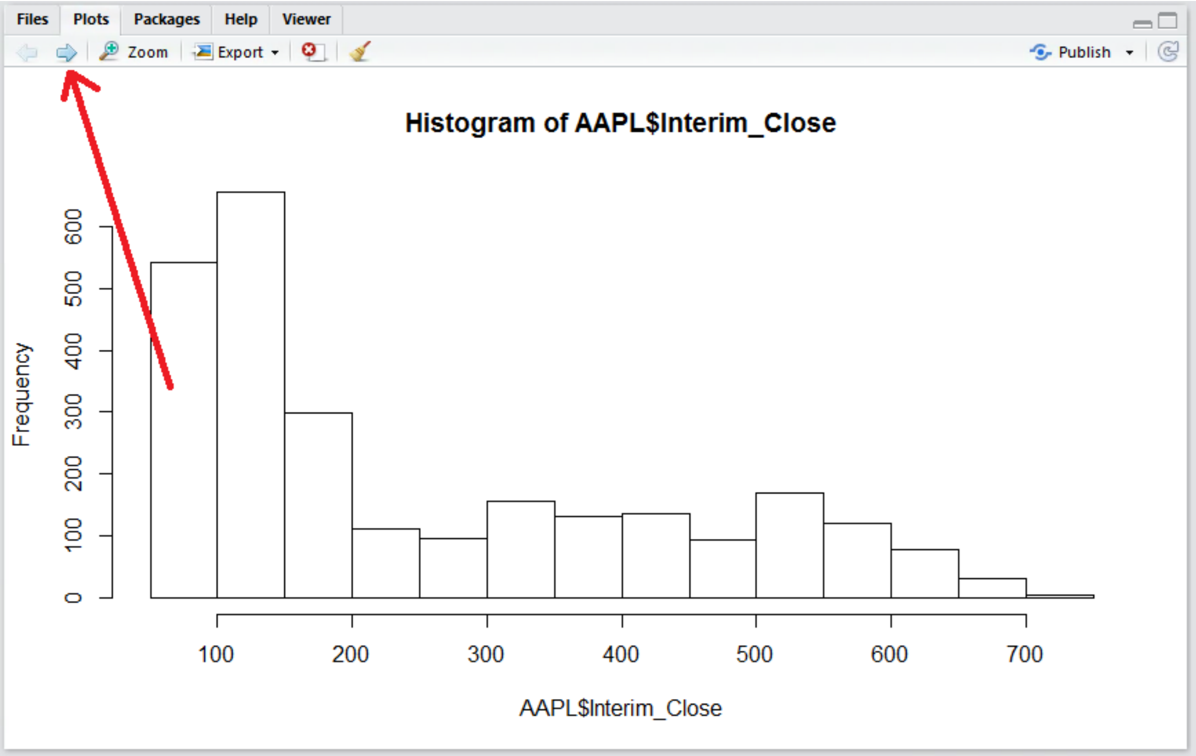
The upper and lower whiskers of the Box Plot represent the minimum and maximum values observed, the upper and lower extremes of the box represent quartile 3 and 1 and lastly the thick horizontal line represents the median. In this example, it can be observed that there is a skew, or compression, towards the lower values.

Procedure 6: Navigate Plots and Export Visualisations.

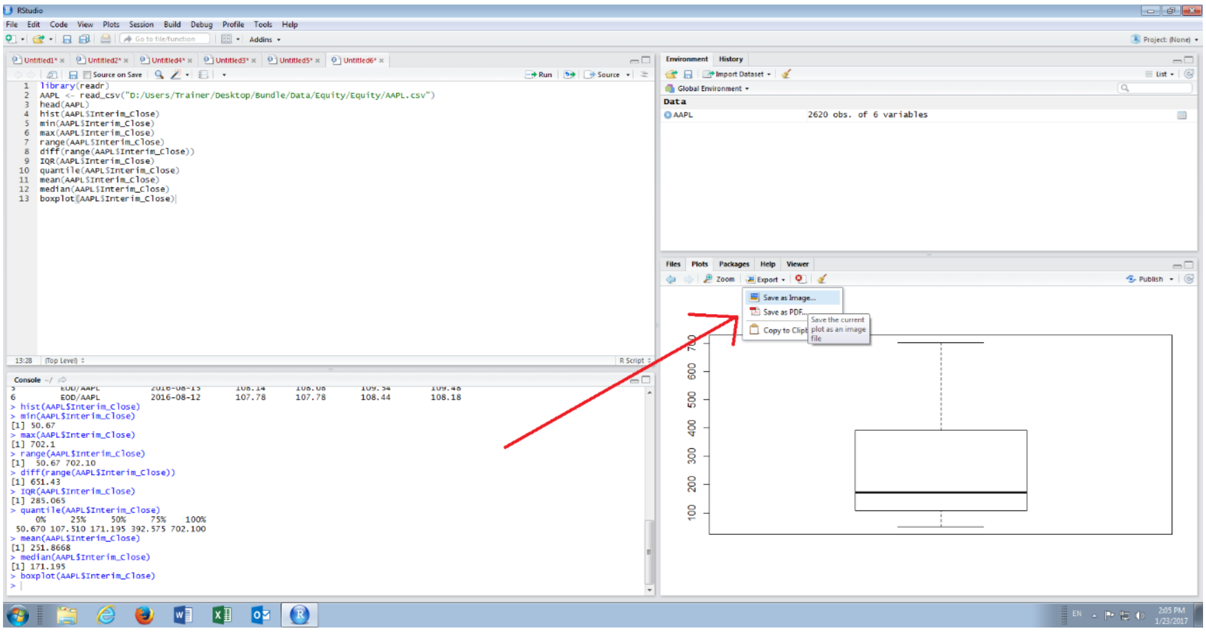
Upon the creation of a box plot at first glance it may appear as if the Histogram created in procedure 55 has been overwritten. Upon closer inspection, it can be seen that this is not the case as there is a back arrow, function, that allows for the paging through plots created:



Clicking on the back arrow will return to the Histogram created in procedure 55:

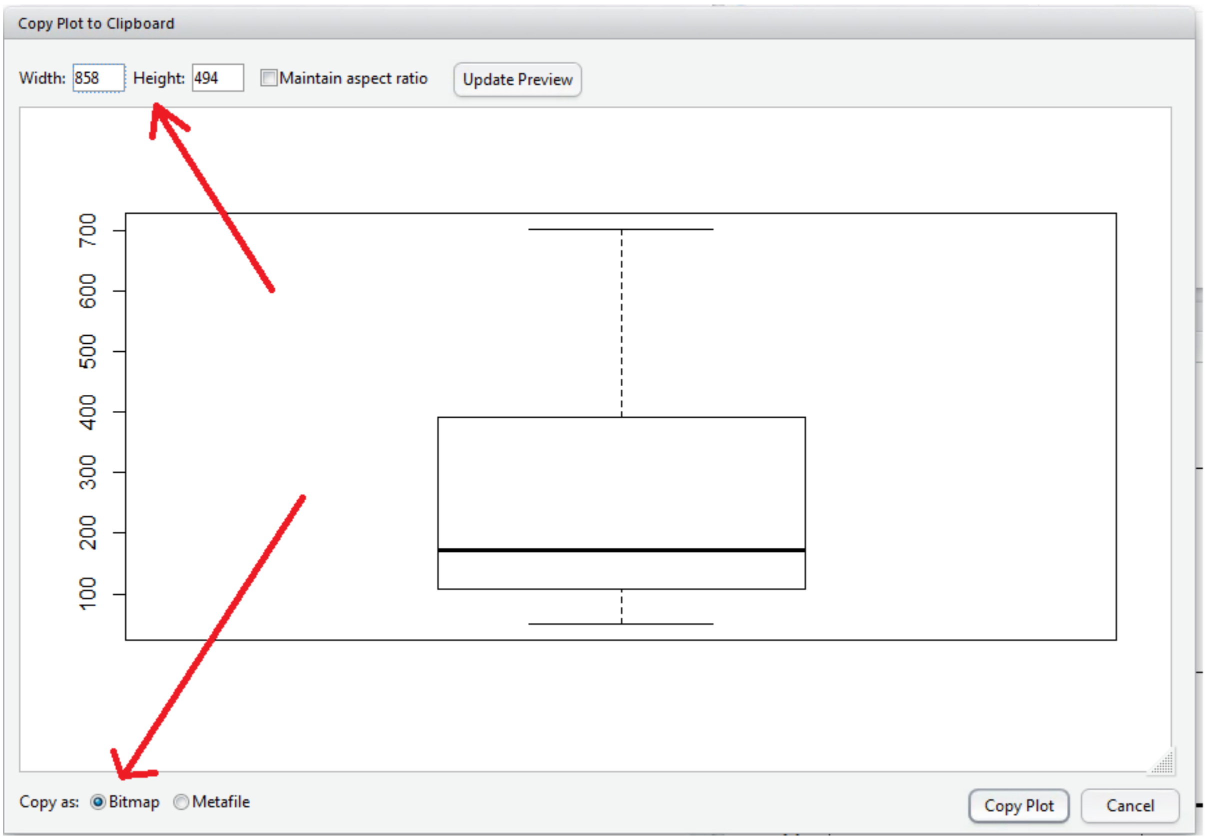


Conversely the forward arrow returns to the newly created Box Plot. RStudio provide a number of mechanisms to export the visualisation via the Export button, clicking on it presents the options:

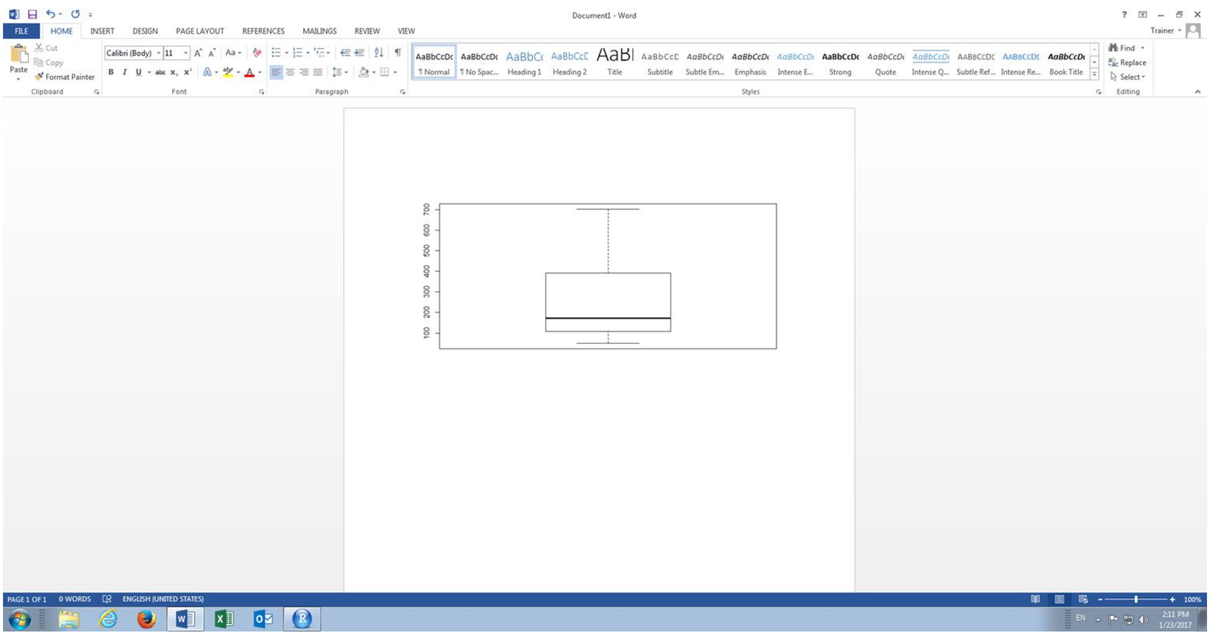


In the drop-down there are several options to export an image from a plot, although the most versatile is to copy the visualisation to clipboard as an image for pasting into a plethora of third party applications, such as Word, via the established Copy \ Paste mechanism familiar to Windows users.

To copy the image, click on the sub menu item Copy to Clipboard which will open a dialog box setting out the specification of the image:



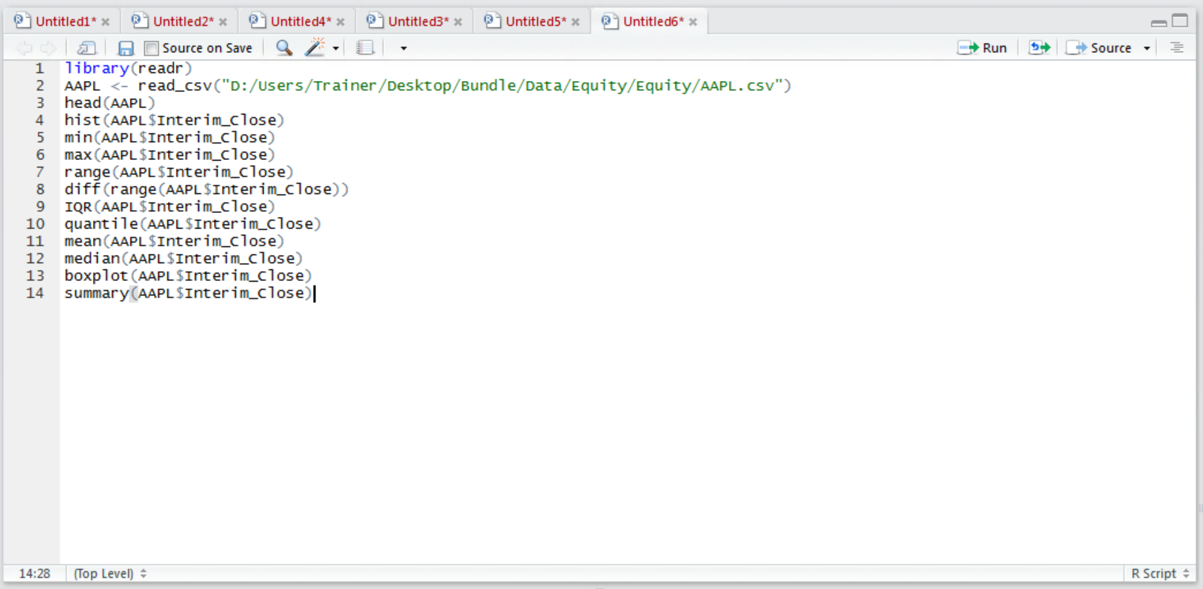
Options for the creation of the image include the dimensions of the image and the precise format \ encoding, in this case the defaults are adequate as a bitmap is a suitably versatile format. Click the Copy Plot button to copy the image to the clipboard. The image can now be pasted into any application that can make use of a bitmap, such as Powerpoint, Word, Excel of Paint:



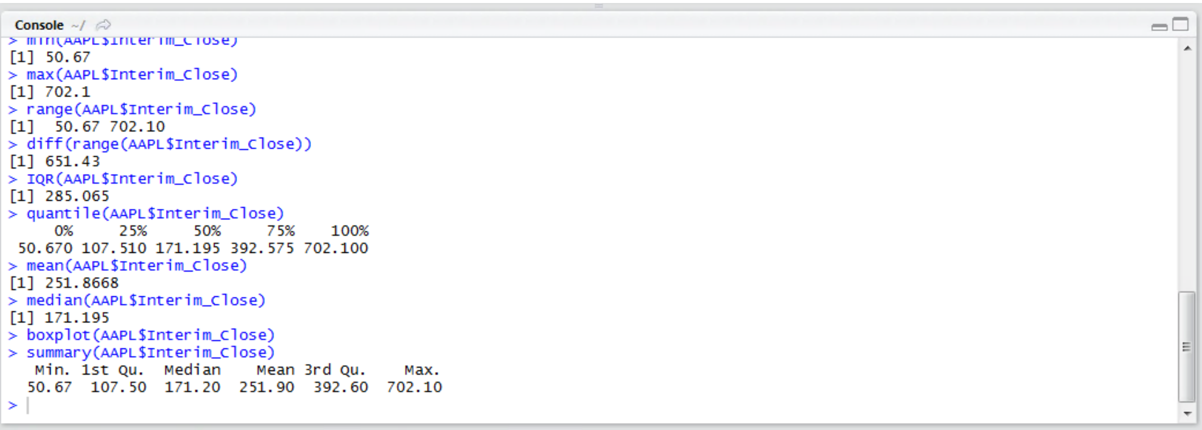
Procedure 7: Create the Variance and Standard Deviation.

The procedures presented in module 4 thus far ignores the existence of a summary() function that produces an analysis of a vector and returns the same summary statistics. To return the summary statistics in this manner type:

summary(AAPL$Interim_Close)



Run the line of script to console:



It can be seen that many of the summary statistics produced one by one are written out to a vector as the result of the summary() function. There is a conspicuous absence of the Variance and Standard Deviation measures in the summary function which calls for the use of the sd() and var() functions. To review the variance of a vector type:

var(AAPL$Interim_Close)

JUBE

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
```

Run the line of script to console:

```
> max(AAPL$Interim_Close)
[1] 702.1
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100% 
 50.670 107.510 171.195 392.575 702.100 
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> boxplot(AAPL$Interim_Close)
> summary(AAPL$Interim_Close)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  50.67  107.50  171.20  251.90  392.60  702.10 
> var(AAPL$Interim_Close)
[1] 31382.2
> 
```

The variance calculation takes the difference between each value and the overall mean, squares it, then takes an average of that. In this case the variance is 3182.2, it could be said that the larger the value the more it varies. The standard deviation, a more useful statistic is simply the square root of the variance. It is more practical to go straight to the Standard Deviation by typing:

sd(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
```

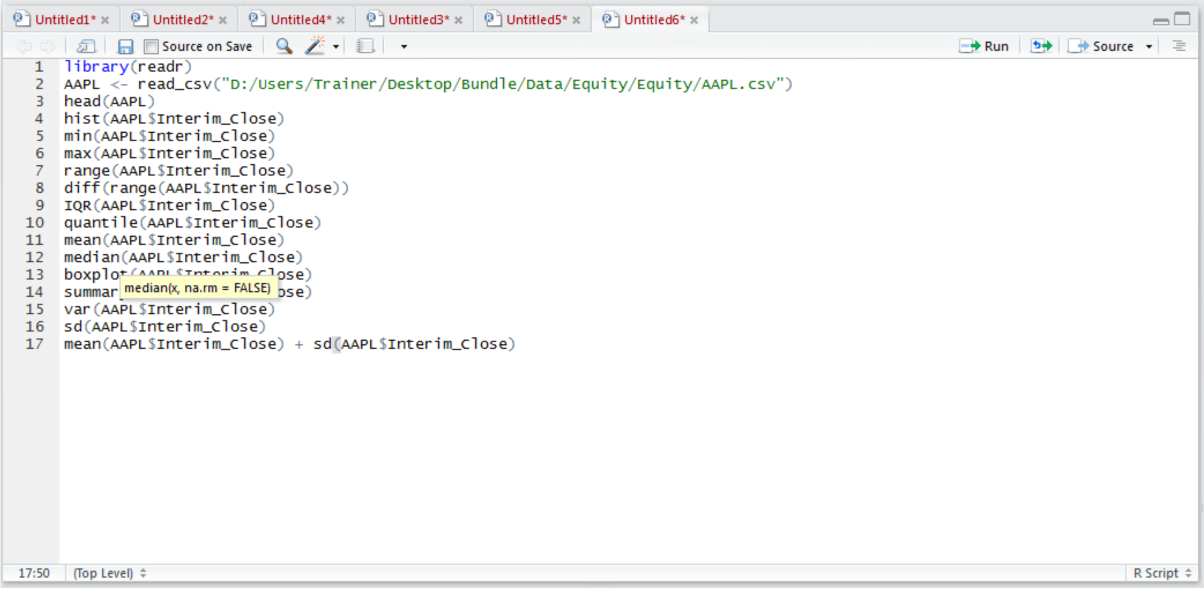
Run the line of script to console:

```
> range(AAPL$Interim_Close)
[1]  50.67 702.10
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> boxplot(AAPL$Interim_Close)
> summary(AAPL$Interim_Close)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  50.67  107.50  171.20  251.90  392.60  702.10
> var(AAPL$Interim_Close)
[1] 31382.2
> sd(AAPL$Interim_Close)
[1] 177.1502
>
```

The standard deviation in this example is 177.1502, a value which has special meaning as adding this to the mean of 251.8668 as produced in procedure 58, it can be said (in a normal distribution at least) that 68.2% of all values will live in the range between 0 (as we can't go below zero) and 429.017. The fact that the lower band is below 0 leads to inference that the distribution is not normally shaped, which is known already from procedure 55, where the vector was plotted to a histogram and box plot.

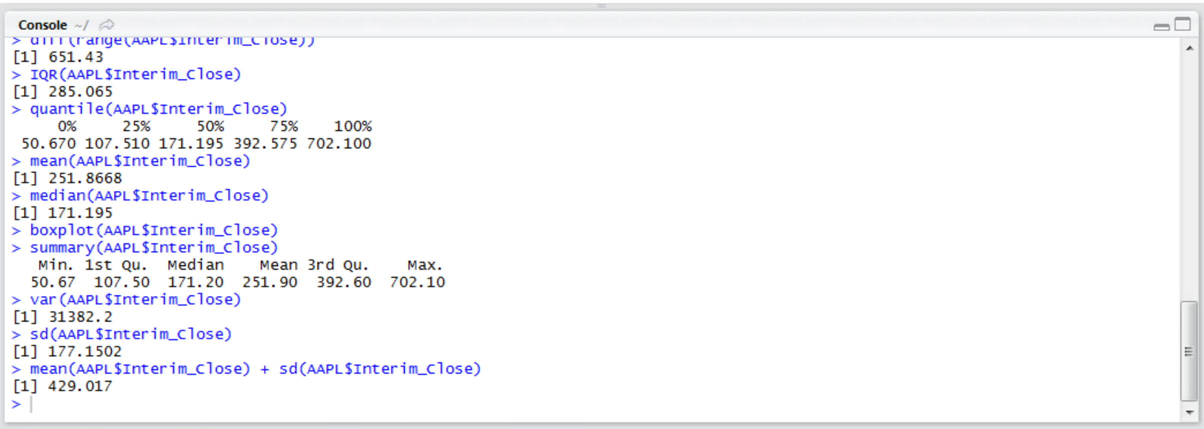
To create an upper band, this being a single Standard Deviation from the Mean:

mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)



```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summar median(x, na.rm = FALSE) ose)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
```

Run the line of script to console:



```
> diff(range(AAPL$Interim_Close))
[1] 651.43
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> boxplot(AAPL$Interim_Close)
> summary(AAPL$Interim_Close)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  50.67  107.50  171.20  251.90  392.60  702.10
> var(AAPL$Interim_Close)
[1] 31382.2
> sd(AAPL$Interim_Close)
[1] 177.1502
> mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
[1] 429.017
>
```

Procedure 8: Calculate a Z Score.

In procedure 61 a calculation was performed representing one standard deviation. A Z Score takes a value then expresses how many standard deviations that value is from the mean. For the purposes of this example, the value to appraise is 201. The formula to calculate how many standard deviations from the mean the value 201 is (201 – Mean) / Standard Deviation.

To identify the Z score of the value 201 type:

(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)

```
Untitled1* × Untitled2* × Untitled4* × Untitled3* × Untitled5* × Untitled6* ×
Source on Save                                   Run   Source
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)

18:58   (Top Level)                                   R Script
```

Run the line of script to console:

```
Console ~/
> IQR(AAPL$Interim_Close)
[1] 285.065
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> boxplot(AAPL$Interim_Close)
> summary(AAPL$Interim_Close)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  50.67  107.50  171.20  251.90  392.60  702.10
> var(AAPL$Interim_Close)
[1] 31382.2
> sd(AAPL$Interim_Close)
[1] 177.1502
> mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
[1] 429.017
> (201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
[1] -0.2871395
>
```

In this example, it can be seen that the value 201 is quite close to the mean being a mere 0.28 standard deviations away from the average. However, as presented in procedures preceding the calculation of the Z score, there are some issue in the way the data is distributed casting some doubt on the relevance of the standard deviation.

Procedure 9: Create a Range Normalisation for a Value.

A useful normalisation is to appraise a value against a scale from the smallest to the largest value. The formula for range normalisation, as in procedure 56 taking the value 201 to be test, is (201 – min) / (max – min) where the minimum and maximum values as calculated as in procedure 56. To test where the value 201 exists on a scale between the minimum and maximum value:

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
```

Run the line of script to console:

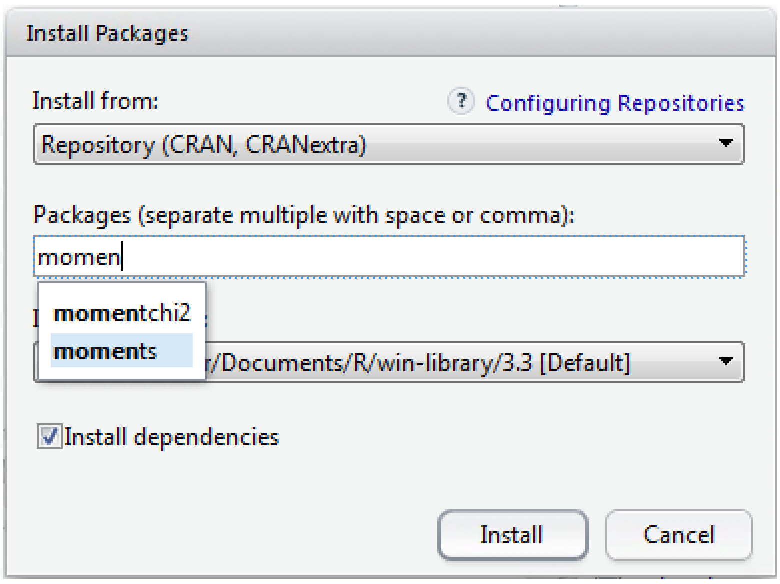
```
> quantile(AAPL$Interim_Close)
     0%     25%     50%     75%    100%
 50.670 107.510 171.195 392.575 702.100
> mean(AAPL$Interim_Close)
[1] 251.8668
> median(AAPL$Interim_Close)
[1] 171.195
> boxplot(AAPL$Interim_Close)
> summary(AAPL$Interim_Close)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  50.67  107.50  171.20  251.90  392.60  702.10
> var(AAPL$Interim_Close)
[1] 31382.2
> sd(AAPL$Interim_Close)
[1] 177.1502
> mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
[1] 429.017
> (201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
[1] -0.2871395
> ((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
[1] 0.2307692
>
```

The output shows that the test value of 201 exists at a point of 23% between the minimum and maximum value observed in the vector.

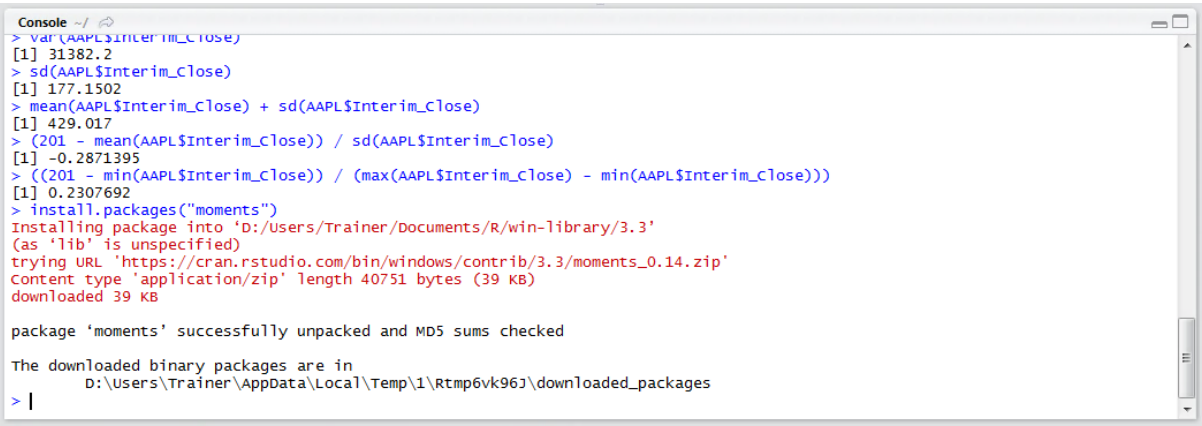
Procedure 10: Create the Skewness and Kurtosis statistics.

It can be observed from procedure 55 that the histogram has a severe lean towards the axis, which would be described as being positively skewed. The positive skew deviating from the shape expected of a normal distribution would be cause mistrust of the standard deviation that was created in procedure 61. Two useful statistics and functions for assessing the extent to which a distribution deviates from the normal distribution is skewness() measuring the lean towards and away from the y axis and kurtosis() measuring how tall or squashed the distribution is.

The functions skewness() and kurtosis() do not exist in the base R packages rather they are available in a package called moments. It follows that the moments package need be installed then loaded. As in procedure 9, search for and install the package moments via RStudio:

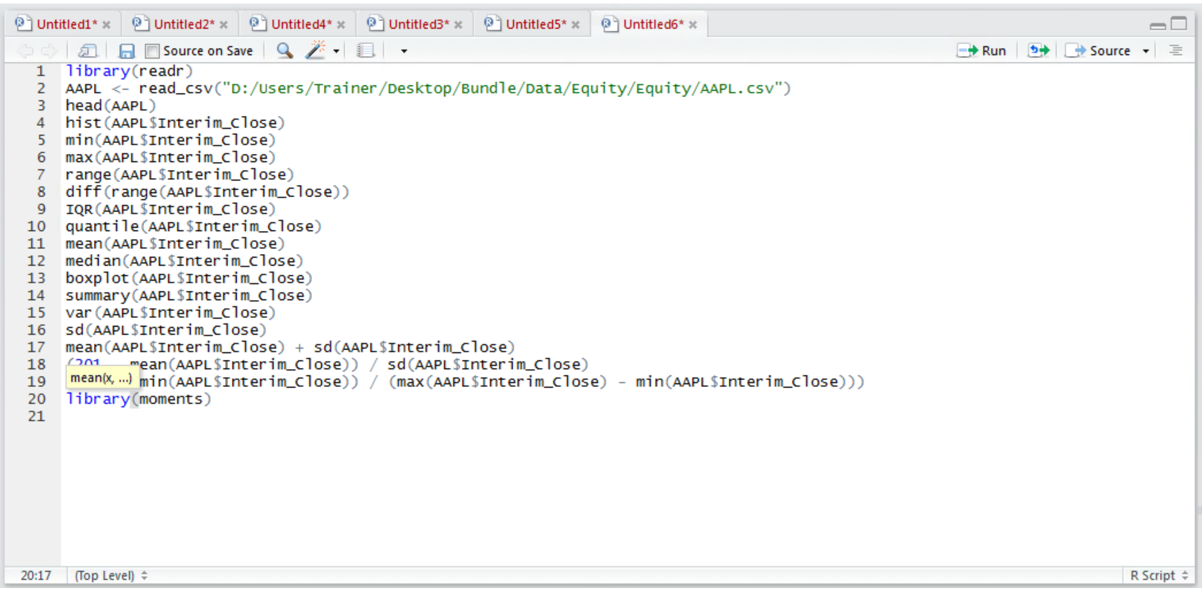


Click the Install button to run the installation instruction to console:



Load the library moments by typing into the script window:

library(moments)



Run the line of script to console:

```
[1] 31382.2
> sd(AAPL$Interim_Close)
[1] 177.1502
> mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
[1] 429.017
> (201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
[1] -0.2871395
> ((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
[1] 0.2307692
> install.packages("moments")
Installing package into ‘D:/Users/Trainer/Documents/R/win-library/3.3’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/moments_0.14.zip'
Content type 'application/zip' length 40751 bytes (39 KB)
downloaded 39 KB

package ‘moments’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\Rtmp6vk96J\downloaded_packages
> library(moments)
>
```

Firstly, in the quest to appraise the extent to which the vector leans towards or away from the axis, type:

skewness(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
library(moments)
skewness(AAPL$Interim_Close)

```

Run the line of script to console:

```
[1] 177.1502
> mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
[1] 429.017
> (201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
[1] -0.2871395
> ((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
[1] 0.2307692
> install.packages("moments")
Installing package into ‘D:/Users/Trainer/Documents/R/win-library/3.3’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/moments_0.14.zip'
Content type 'application/zip' length 40751 bytes (39 KB)
downloaded 39 KB

package ‘moments’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\Rtmp6vk96J\downloaded_packages
> library(moments)
> skewness(AAPL$Interim_Close)
[1] 0.8176304
>
```

It can be observed that there is a positive value returned, indicating that there is indeed lean and owing to it being positive, that the lean is towards the y axis (which is of course what was visually observed in procedure 55). Secondly to understand if the distribution is tall or squat, verify the kurtosis by typing:

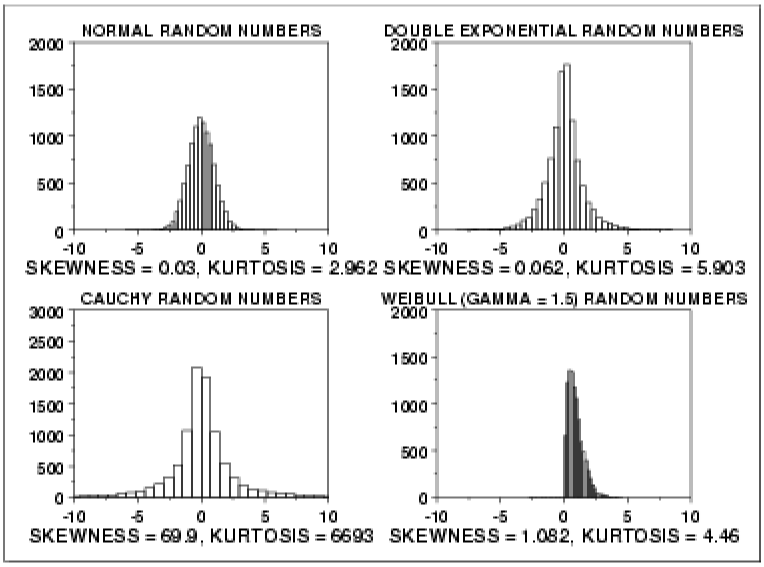
kurtosis(AAPL$Interim_Close)

```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
library(moments)
skewness(AAPL$Interim_Close)
kurtosis(AAPL$Interim_Close)
```

Run the line of script to console:

```
[1] 429.017
> (201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
[1] -0.2871395
> ((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
[1] 0.2307692
> install.packages("moments")
Installing package into ‘D:/Users/Trainer/Documents/R/win-library/3.3’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/moments_0.14.zip'
Content type 'application/zip' length 40751 bytes (39 KB)
downloaded 39 KB

package ‘moments’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\Rtmp6vk96J\downloaded_packages
> library(moments)
> skewness(AAPL$Interim_Close)
[1] 0.8176304
> kurtosis(AAPL$Interim_Close)
[1] 2.275733
>
```

The kurtosis is a difficult statistic to make sense of and in many respects the skewness is a more useful statistic. To make an assessment of the shape of the distribution, typically, all summary statistics need to be considered:

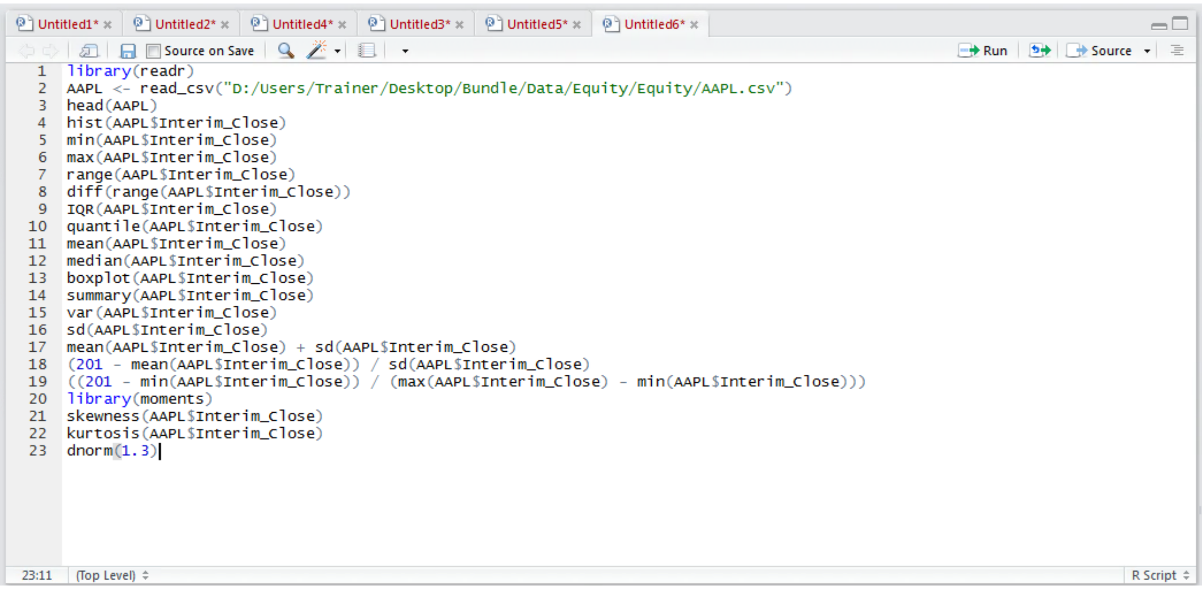


Procedure 11: Create Probabilities from a test value in a normal distribution.

One of the useful properties of a normal distribution is the ability to predict the probability of that value occurring. Intuitively values on either end of the tail would seem to be extremely unlikely to happen and functions in R can facilitate the creation of a probability to express this. In this procedure there are two functions that will be used to gain a sense for the probability of a particular value occurring dnorn() and pnorm() both taking the z score (the number of standard deviations away from the mean) as their arguments.

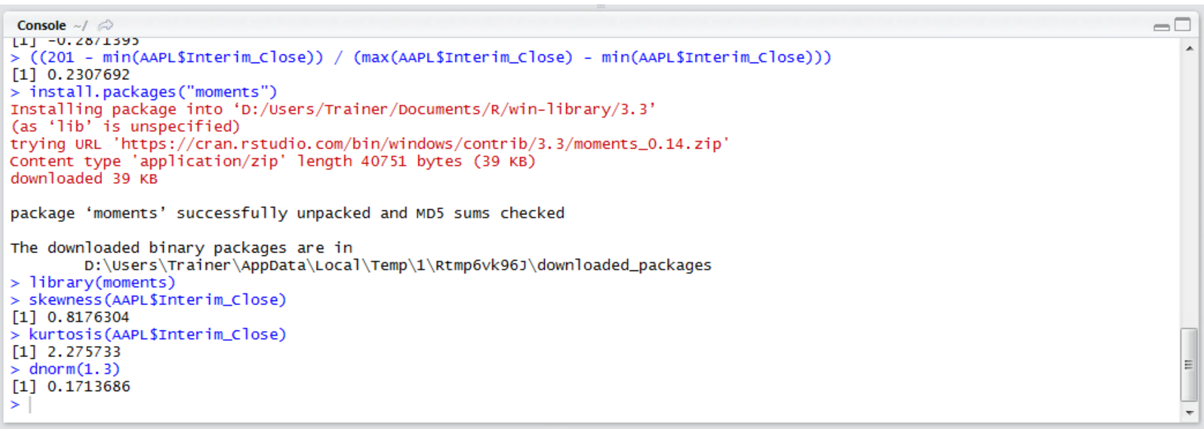
The dnorm() function returns the position of the value on the y axis, which has certain predictive properties when overlaid on a histogram created as per procedure 55. Taking a value of 1.3 standard deviations from the average and returning the approximate height of the point in the y axis type:

dnorm(1.5)



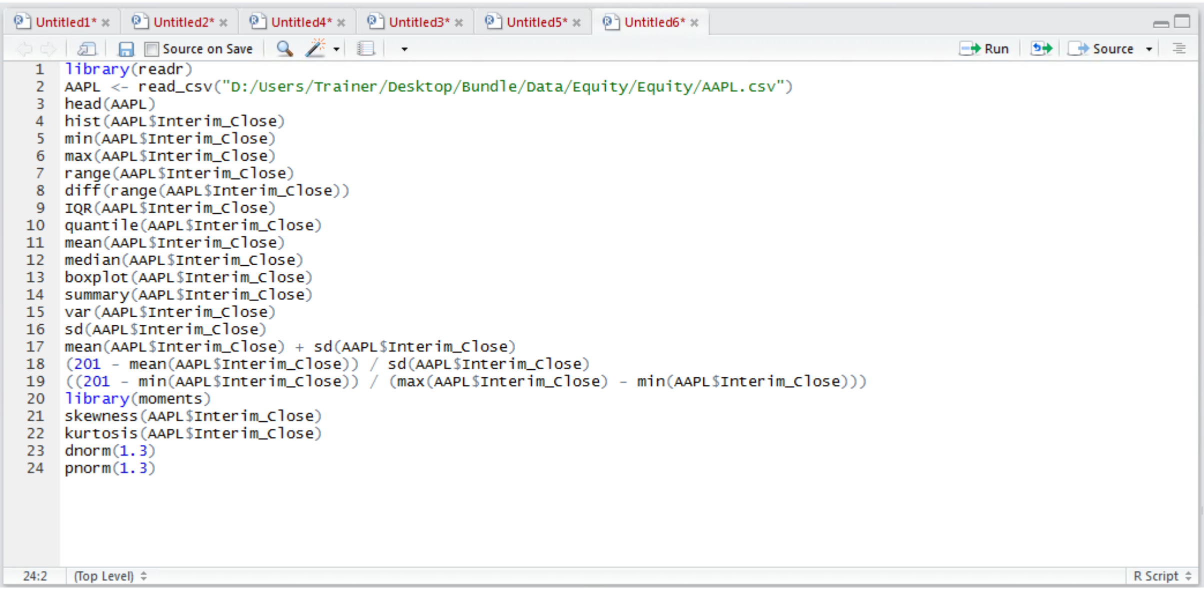
```
library(readr)
AAPL <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Equity/AAPL.csv")
head(AAPL)
hist(AAPL$Interim_Close)
min(AAPL$Interim_Close)
max(AAPL$Interim_Close)
range(AAPL$Interim_Close)
diff(range(AAPL$Interim_Close))
IQR(AAPL$Interim_Close)
quantile(AAPL$Interim_Close)
mean(AAPL$Interim_Close)
median(AAPL$Interim_Close)
boxplot(AAPL$Interim_Close)
summary(AAPL$Interim_Close)
var(AAPL$Interim_Close)
sd(AAPL$Interim_Close)
mean(AAPL$Interim_Close) + sd(AAPL$Interim_Close)
(201 - mean(AAPL$Interim_Close)) / sd(AAPL$Interim_Close)
((201 - min(AAPL$Interim_Close)) / (max(AAPL$Interim_Close) - min(AAPL$Interim_Close)))
library(moments)
skewness(AAPL$Interim_Close)
kurtosis(AAPL$Interim_Close)
dnorm(1.3)
```

Run the line of script to console:

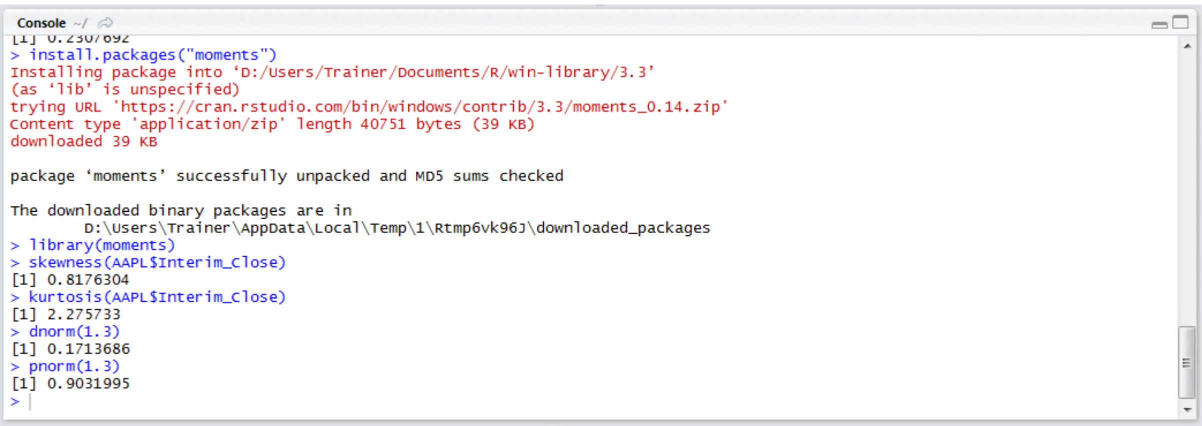


A far more useful measure is of cumulative probability which, when knowing a z score, expresses the percentage probability that the value would fall somewhere below that Z score. To obtain the cumulative probability of a value having a Z score of 1.3 being less than that value type:

pnorm(1.5)



Run the line of script to console:



It follows that this Z score and values up to and including this z score are around 90% certain.

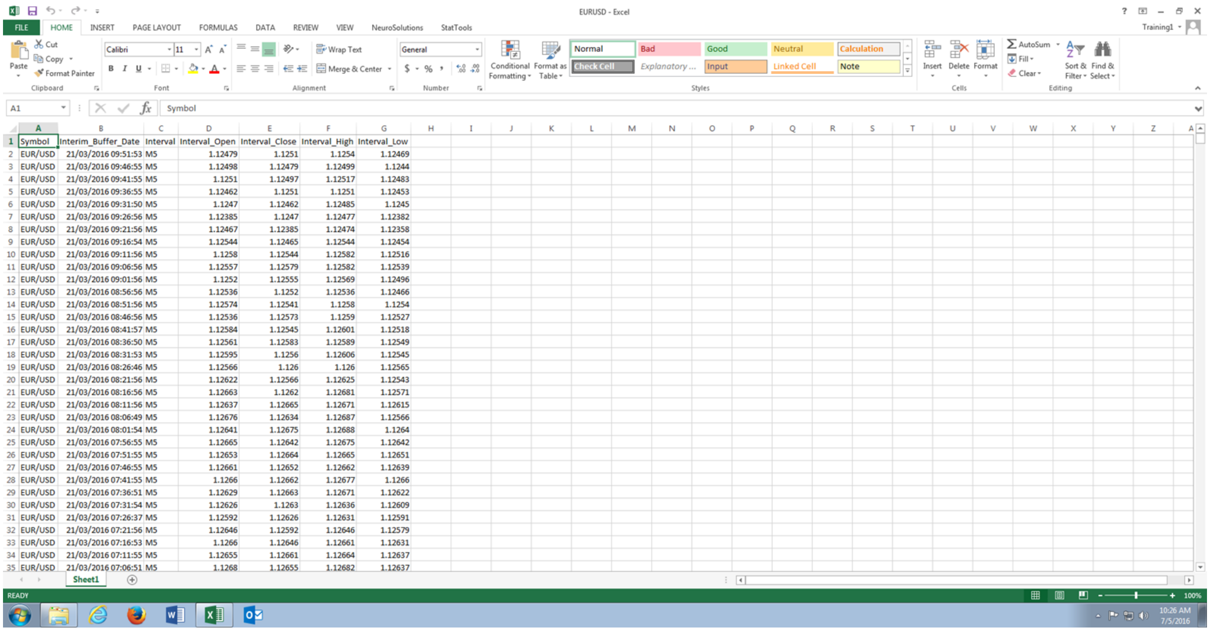
Procedure 12: Create a Log Transformation.

Procedure 13: Reverse a Log Transformation.

Module 6: Abstraction and Transformations

Abstraction and Transformation is the process of creating pseudo \ derived columns in a spreadsheet based upon behavioural characteristics, in this example of a financial instrument prices observed over time. The example spreadsheet, \Training\Data\FX\EURUSD.csv, is ordered from the newest example through to the oldest example, which is an assumption made for execution of the following procedures.

In the same manner as Module 2, open the spreadsheet \Training\Data\FX\EURUSD.csv.



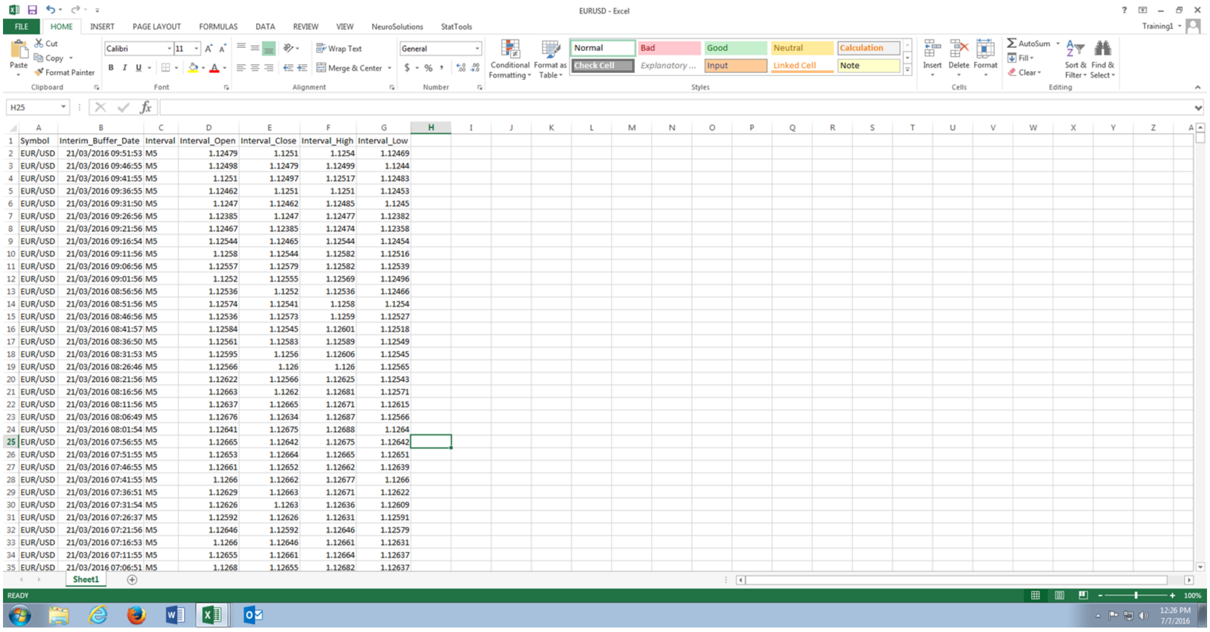
Procedure 1: Create a Dependent Variable in Time Series Data.

Creating a dependent variable in an ordered time series dataset is a matter of agreeing a horizon, in this case two hours, then looking for the first record in the dataset where that two hours would be fully completed.

The example dataset is formed of five minute intervals, and so, it would render only the 24th example in the dataset as being complete. There a number of ways to determine an independent variable in time series data, for example the value of interest may be the price at the horizon or some summary measure of all prices observed in that horizon (e.g. Mean).

In this example, the price AT the Horizon is the dependent variable.

Find the example where there are 24 examples ahead \ in front or the example to be predicted (which in a five-minute interval would represent two hours). In this example, taking into account the header, the first record where there are 24 examples in front is row 25:

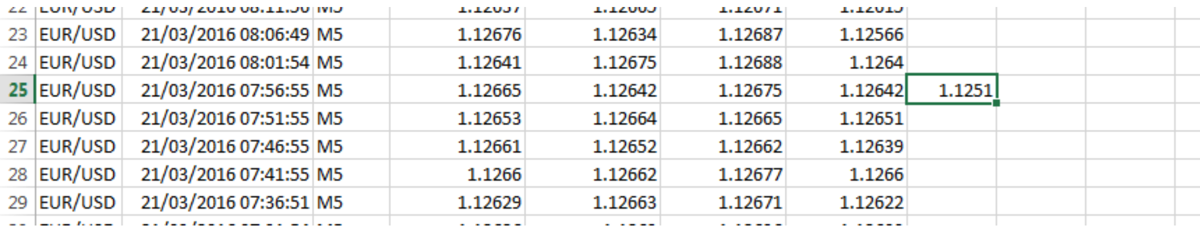


Select the cell adjacent to the last field in the example and set the formula to reference the Interval_Close, some 24 examples forward, in cell E2:

SUM =E2

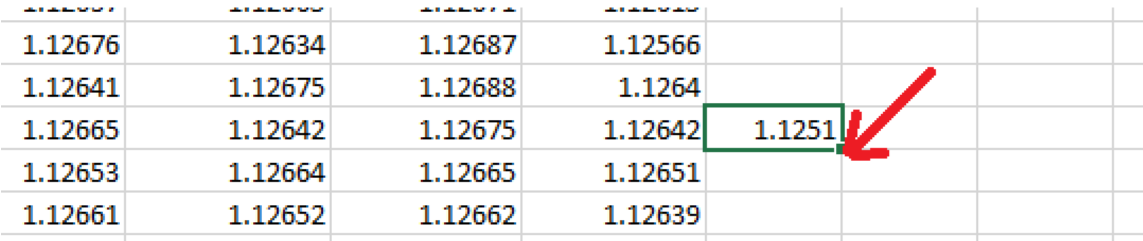
	A	B	C	D	E	F	G	H	I
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low		
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469		
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244		
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483		
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453		
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245		
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382		
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358		
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454		
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516		
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539		
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496		
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466		
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254		
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527		
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518		
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549		
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545		
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565		
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543		
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571		
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615		
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566		
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264		
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	=E2	
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651		
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639		

Commit the formula, keeping the cell in focus \ selected:



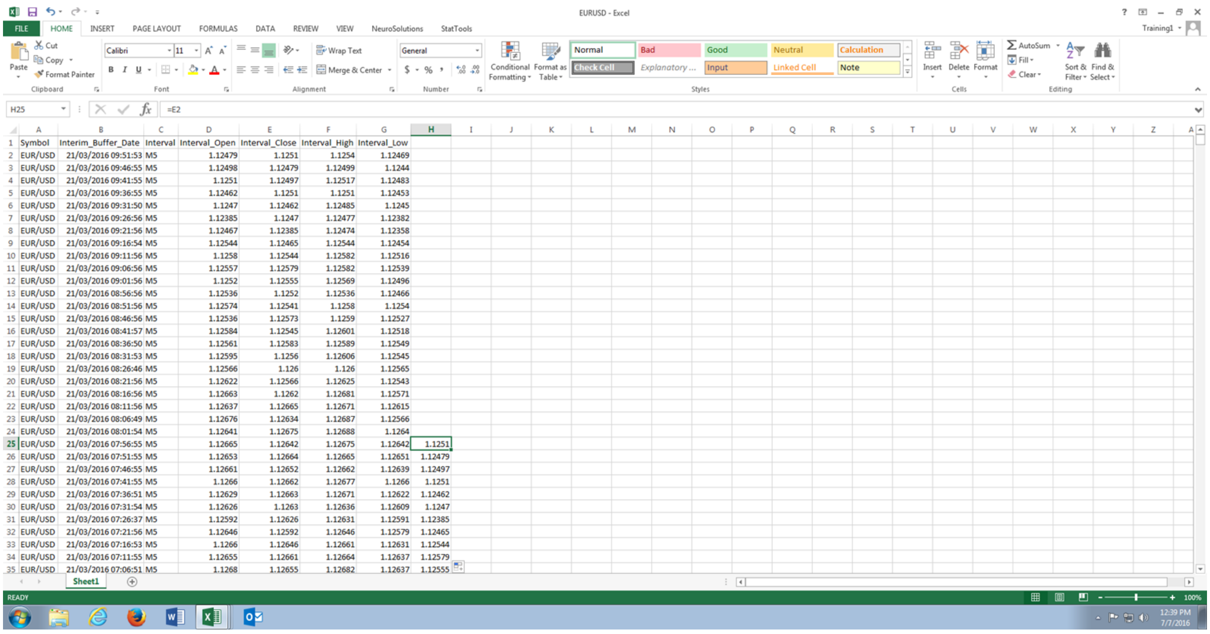
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566	
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264	
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	

Perform the procedure to Auto Fill the formula down for the remainder of the records. To invoke autofill, hover on the selected cell in the extreme bottom right corner of the selected cell until the crosshairs are shown:



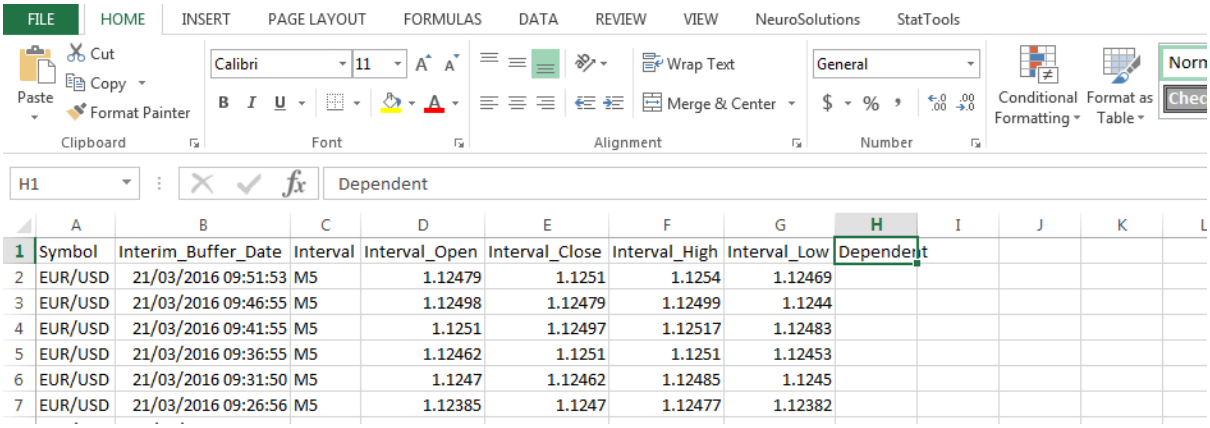
1.12676	1.12634	1.12687	1.12566	
1.12641	1.12675	1.12688	1.1264	
1.12665	1.12642	1.12675	1.12642	1.1251
1.12653	1.12664	1.12665	1.12651	
1.12661	1.12652	1.12662	1.12639	

A double click on the cross hair will replicate the formula for the remainder of the examples in the dataset, while maintaining the same step (for example E2, will step to E3 for example 26) and so on:



The procedure of filling down in this manner will be used extensively in subsequent procedures and is referred simply as 'Fill Down' herein.

For completeness, ensure that the dependent variable is given a header, in this case called 'Dependent'. Click on the very first row of the spreadsheet and the cell which would represent the header, in this case D1, enter the header name:



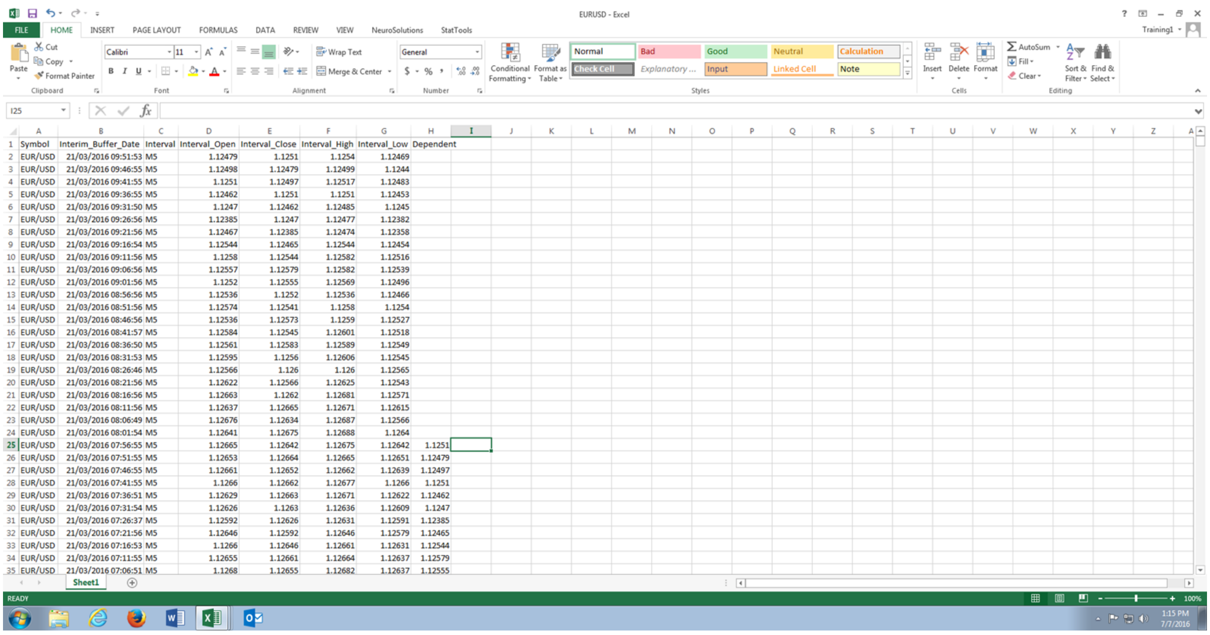
	A	B	C	D	E	F	G	H
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Dependent
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469	
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244	
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483	
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453	
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245	
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382	

The procedure of naming a column in this manner will be used extensively in subsequent procedures and is referred simply as 'Name..' herein.

Procedure 2: Create an Independent variable based on a basic summary statistic.

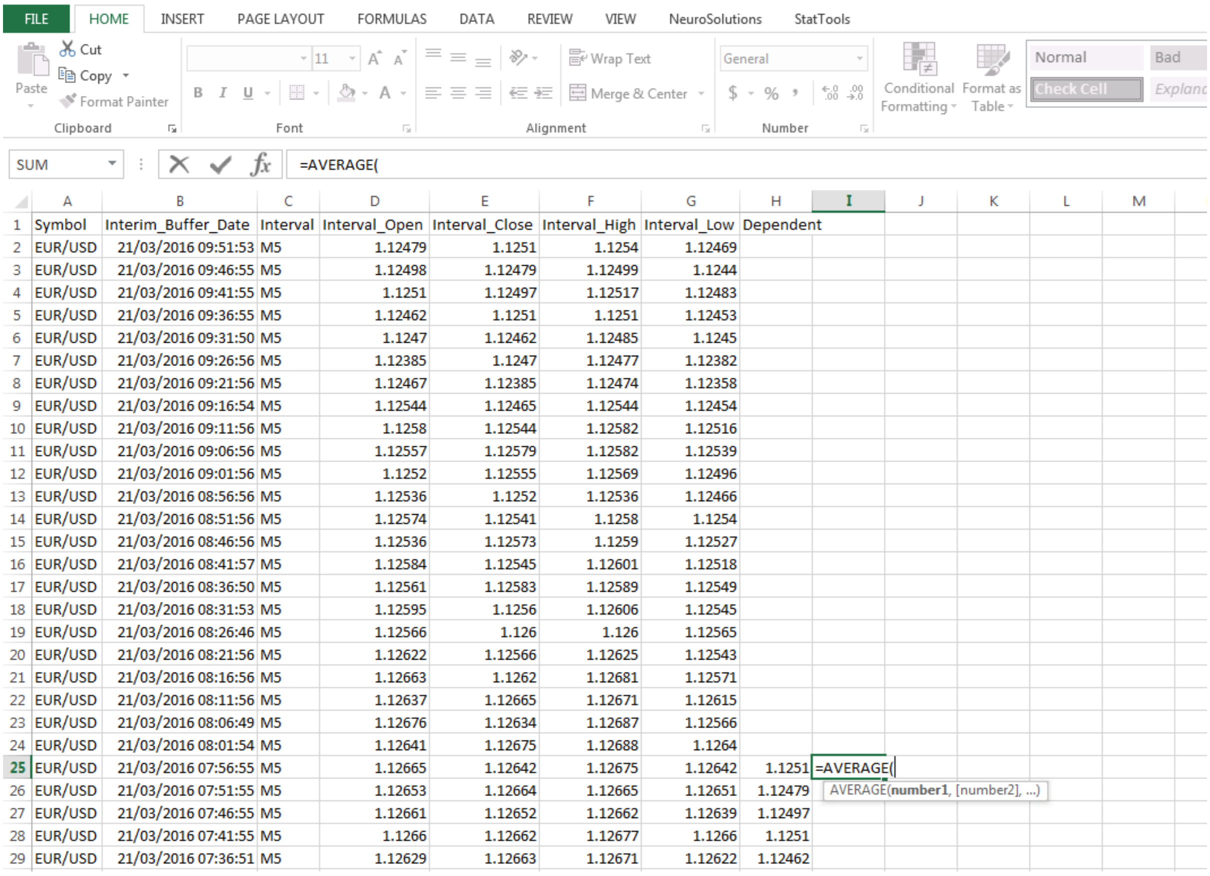
The procedure to create an independent variable is similar to the procedure of creating a dependent variable, except for the concept of Horizon (how far forward) is replaced with the concept of Scope (how far backwards into the historic exemplars). In this example, the Horizon is focusing on 24 intervals forward, where the Scope will be 700 intervals backwards.

For the first example where there is a Dependent Variable calculated, in this case H25, click on the cell immediately to the right, in this case I25. It follows that the Independent Variable will be a column right adjacent to the Dependent variable:



Begin typing the Excel function to be used in aggregation, in this case AVERAGE:

=AVERAGE(

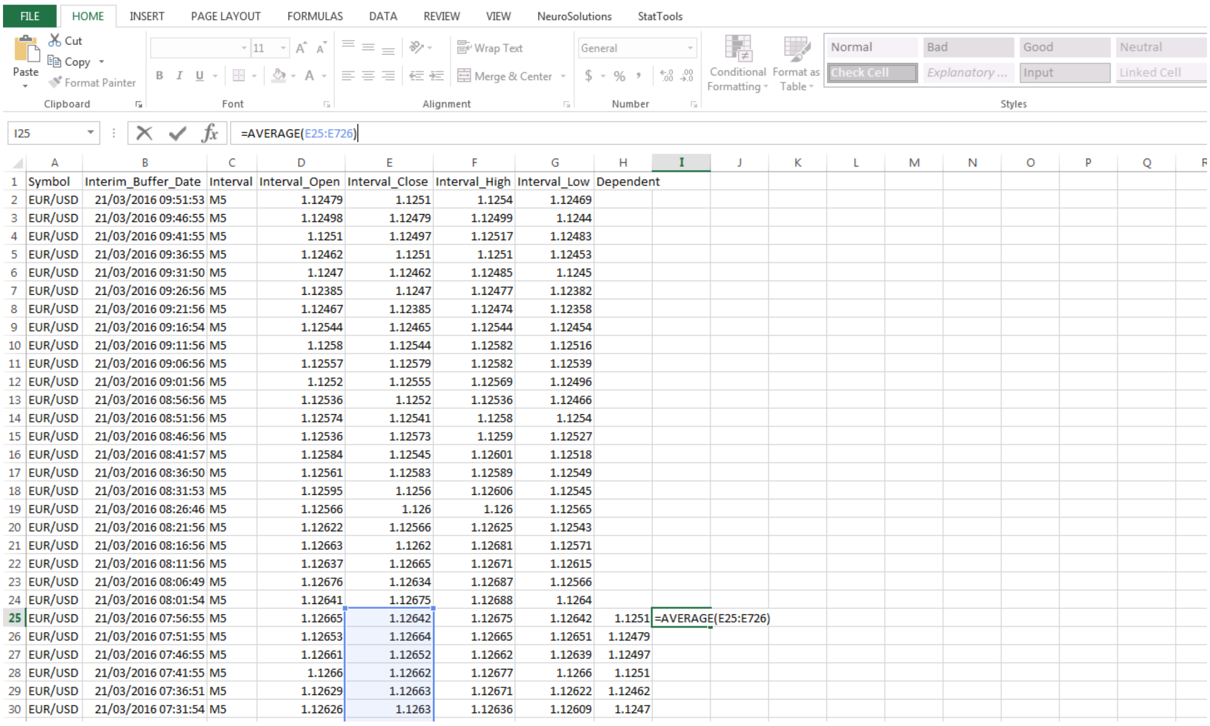


	A	B	C	D	E	F	G	H	I
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Dependent	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469		
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244		
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483		
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453		
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245		
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382		
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358		
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454		
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516		
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539		
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496		
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466		
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254		
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527		
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518		
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549		
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545		
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565		
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543		
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571		
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615		
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566		
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264		
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	=AVERAGE(
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	

In this example, the scope is the last 700 intervals backwards. Therefore, taking the Interval_Close column, which is Colum E and excluding the first 24 examples which are not complete, the scope can be described as being all cells in column F between 25 to 726, or rather:

E25:E726

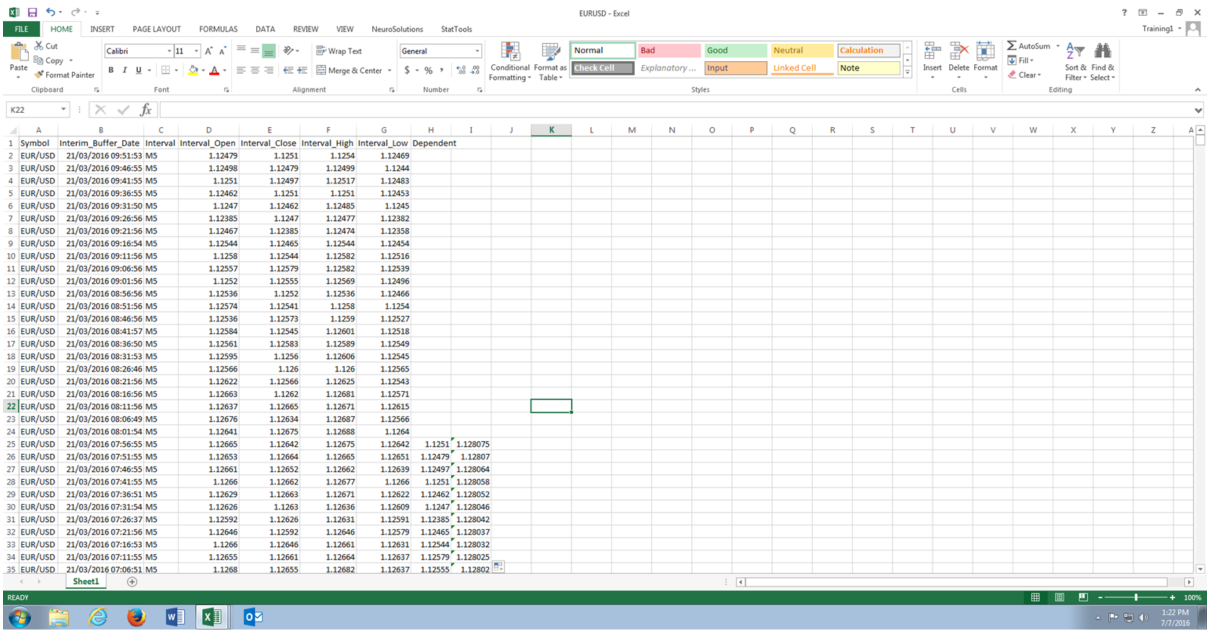
Complete the formula with the range E25:E726, closing the parenthesis:



=AVERAGE(E25:E726)

	A	B	C	D	E	F	G	H	I
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Dependent	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469		
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244		
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483		
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453		
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245		
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382		
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358		
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454		
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516		
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539		
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496		
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466		
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254		
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527		
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518		
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549		
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545		
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565		
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543		
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571		
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615		
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566		
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264		
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	=AVERAGE(E25:E726)
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	
30	EUR/USD	21/03/2016 07:31:54	M5	1.12626	1.1263	1.12636	1.12609	1.1247	

Commit the formula in Excel, then Fill Down:



	A	B	C	D	E	F	G	H	I
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Dependent	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469		
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244		
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483		
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453		
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245		
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382		
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358		
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454		
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516		
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539		
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496		
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466		
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254		
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527		
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518		
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549		
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545		
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565		
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543		
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571		
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615		
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566		
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264		
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052
30	EUR/USD	21/03/2016 07:31:54	M5	1.12626	1.1263	1.12636	1.12609	1.1247	1.128046
31	EUR/USD	21/03/2016 07:26:37	M5	1.12592	1.12626	1.12631	1.12591	1.12385	1.128042
32	EUR/USD	21/03/2016 07:21:56	M5	1.12646	1.12592	1.12646	1.12579	1.12465	1.128037
33	EUR/USD	21/03/2016 07:16:53	M5	1.1266	1.12646	1.12661	1.12631	1.12544	1.128032
34	EUR/USD	21/03/2016 07:11:55	M5	1.12655	1.12661	1.12664	1.12637	1.12579	1.128025
35	EUR/USD	21/03/2016 07:06:51	M5	1.1268	1.12655	1.12682	1.12637	1.12555	1.12802

Name the Independent Variable:

Average_700

	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_700
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469		
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244		
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483		
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453		
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245		
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382		
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358		
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454		
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516		
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539		
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496		
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466		
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254		
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527		
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518		
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549		
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545		
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565		
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543		
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571		
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615		
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566		
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264		
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807

This procedure can be used in any of the aggregation functions available in Excel, of which the following concepts have been introduced in Module 2 and are described below with their Excel counterparts:

- Max = MAX
- Min = MIN
- Mean= AVERAG
- Std. Dev = STDEV
- Mode = MODE
- Interquartile Range = QUARTILE
- Range = MAX = MIN
- Skew = SKEW
- Kurtosis = KURT
- Sum = SUM
- Median = Median

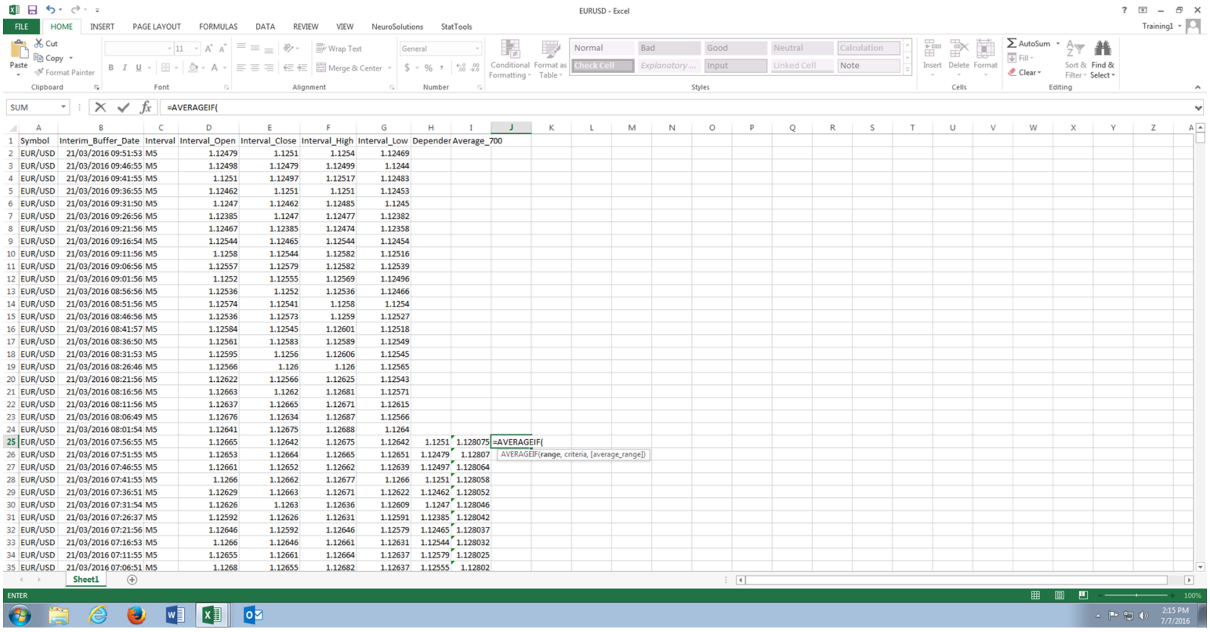
The process of Abstraction would typically rely on a creative and varied use of all of these functions across a varying Scope (the intervals backwards, in this case 700 intervals).

Procedure 3: Create an Independent Variable based on threshold aggregation.

As only a slight variation on Procedure 9, which introduced the concept of creating an independent variable with no reference to the current Interval_Close, this procedure sets about creating a variable that makes a reference to the current Interval_Close and using this as filtering for the aggregation.

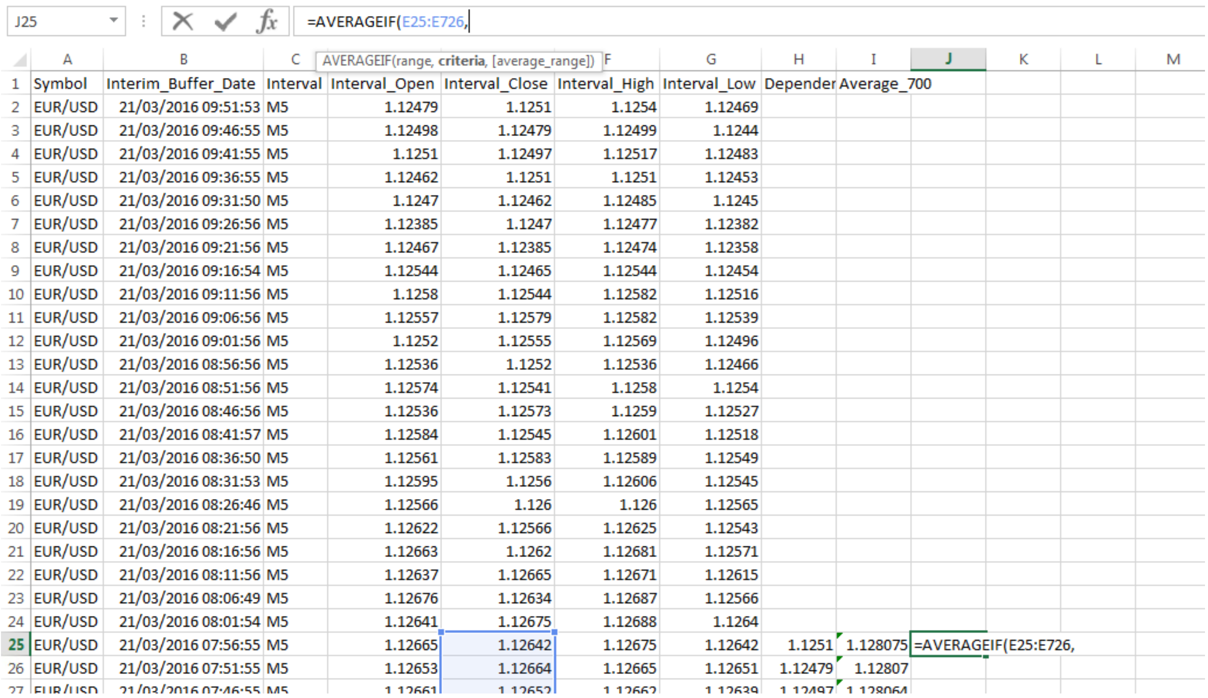
Start by creating a new independent variable in the same manner as Procedure 9, however instead of using =AVERAGE, the AVERAGEIF function is going to be used. Begin the function as:

=AVERAGEIF(



Specify the range parameter as the same scope as that used in Procedure 9, being the last 700 intervals backwards from the current example:

E25:E726



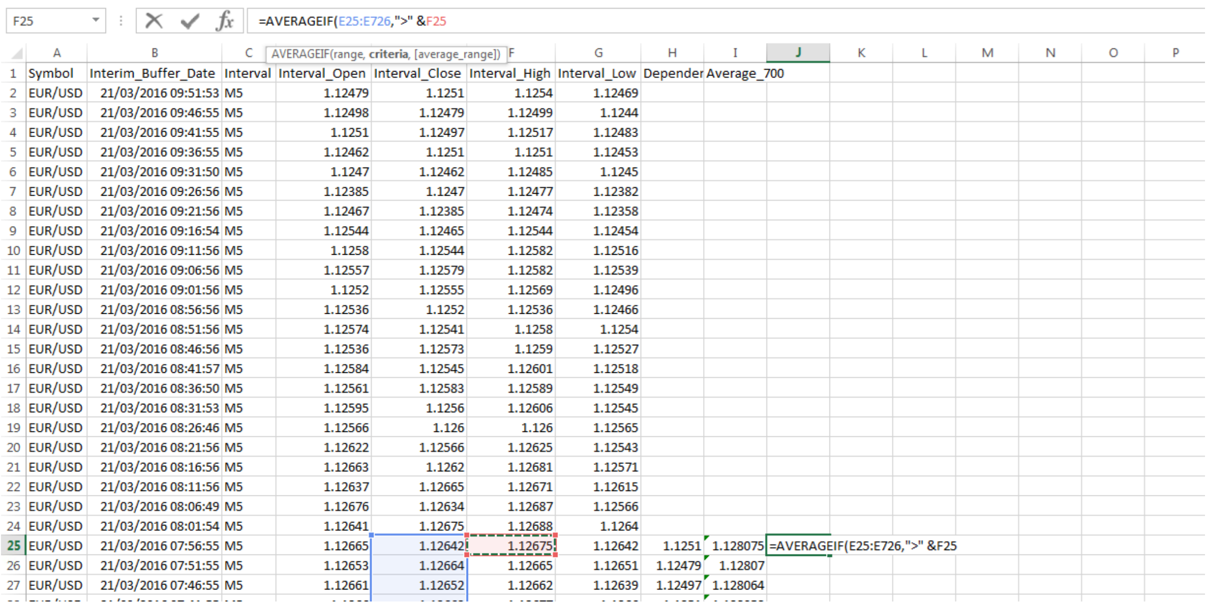
	A	B	C	D	E	F	G	H	I	J
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_700	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469			
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244			
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483			
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453			
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245			
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382			
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358			
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454			
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516			
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539			
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496			
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466			
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254			
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527			
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518			
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549			
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545			
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565			
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543			
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571			
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615			
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566			
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264			
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	=AVERAGEIF(E25:E726,
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	

The next parameter of the AVERAGEIF is the string representing the filter. This string will be a concatenation which will include a condition and the cell value of the Interval_Close, as cell E25, constructed as follows:

“>” & E25

Therefore, completing the parameter in the function:

=AVERAGEIF(E25:E726,">" &F25

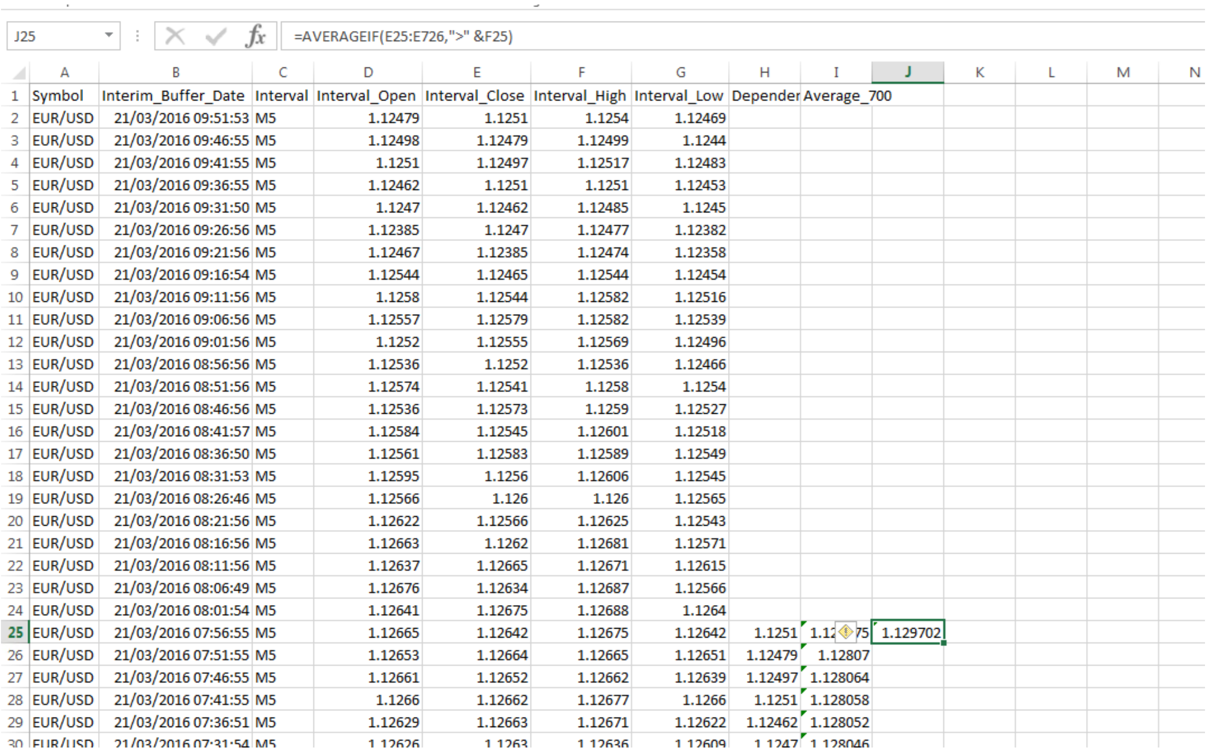


F25 =AVERAGEIF(E25:E726,">" &F25

AVERAGEIF(range, criteria, [average_range])

	A	B	C	D	E	F	G	H	I	J
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_700	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469			
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244			
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483			
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453			
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245			
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382			
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358			
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454			
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516			
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539			
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496			
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466			
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254			
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527			
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518			
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549			
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545			
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565			
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543			
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571			
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615			
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566			
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264			
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	=AVERAGEIF(E25:E726,">" &F25
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	

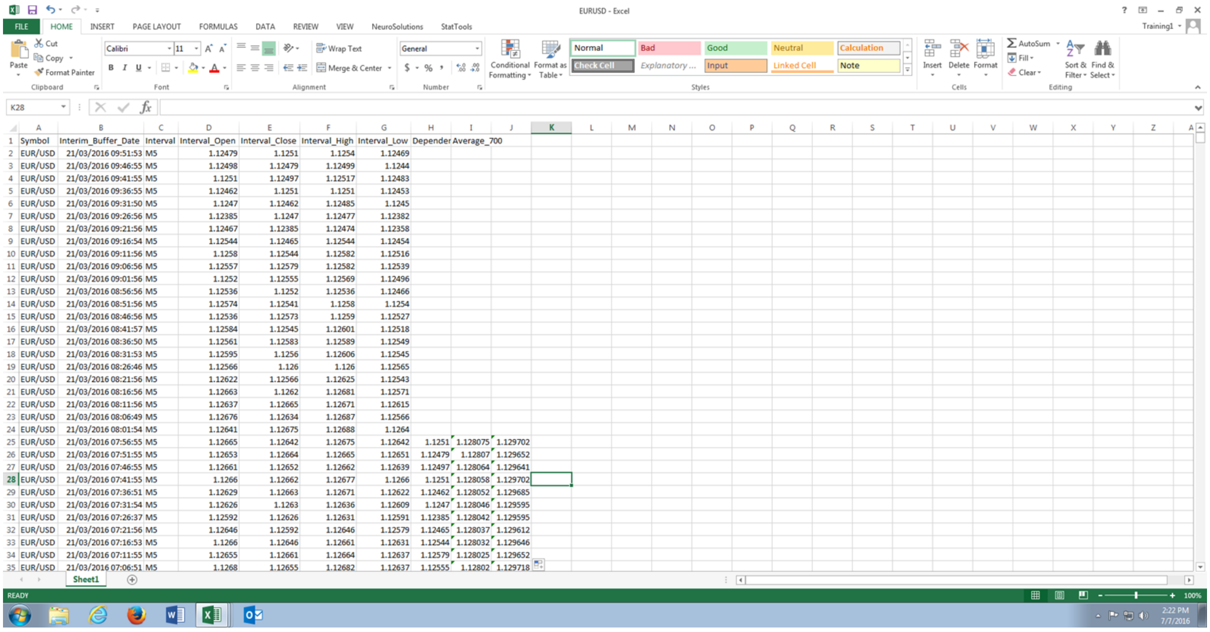
Close the parenthesis and commit the formula:



J25 =AVERAGEIF(E25:E726,">" &F25)

	A	B	C	D	E	F	G	H	I	J
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_700	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469			
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244			
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483			
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453			
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245			
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382			
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358			
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454			
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516			
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539			
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496			
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466			
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254			
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527			
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518			
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549			
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545			
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565			
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543			
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571			
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615			
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566			
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264			
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.12[illegible]75	1.129702
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058	
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052	
30	EUR/USD	21/03/2016 07:31:54	M5	1.12626	1.1263	1.12636	1.12609	1.1247	1.128046	

Fill down to ensure that the remaining examples are given the values of this independent variable:



The most commonly used conditional aggregation functions would be:

- COUNTIF
- AVERAGEIF
- SUMIF

The process of Abstraction would typically rely on a creative and varied use of all of these functions across a varying Scope (the intervals backwards, in the case of this procedure 700) and threshold, which can be anchored to the reference (as in this example) or another Independent Variable that has been horizontally abstracted.

Procedure 4: Creating a Ratio Independent Variable in Horizontal Abstraction.

With extensive abstraction having taken place using summary statistics or filtered aggregation across the Scope, this procedure looks to extend these variables, bringing them together in ratios. Ratios are a method to normalise data and typically make the analysis more useful for linear modelling techniques.

In this example, we are going to represent the average price observed over the scope and compare that as a ratio to the current \ prevailing Interval_Close. Create a new Independent Variable in the same manner as preceding procedures, in this case selecting cell K25 as the starting point. To create a ratio between the variables, simply divide one variable into the next, for example the Average_700 divided by the current \ prevailing Interval_Close as contained in cell E25:

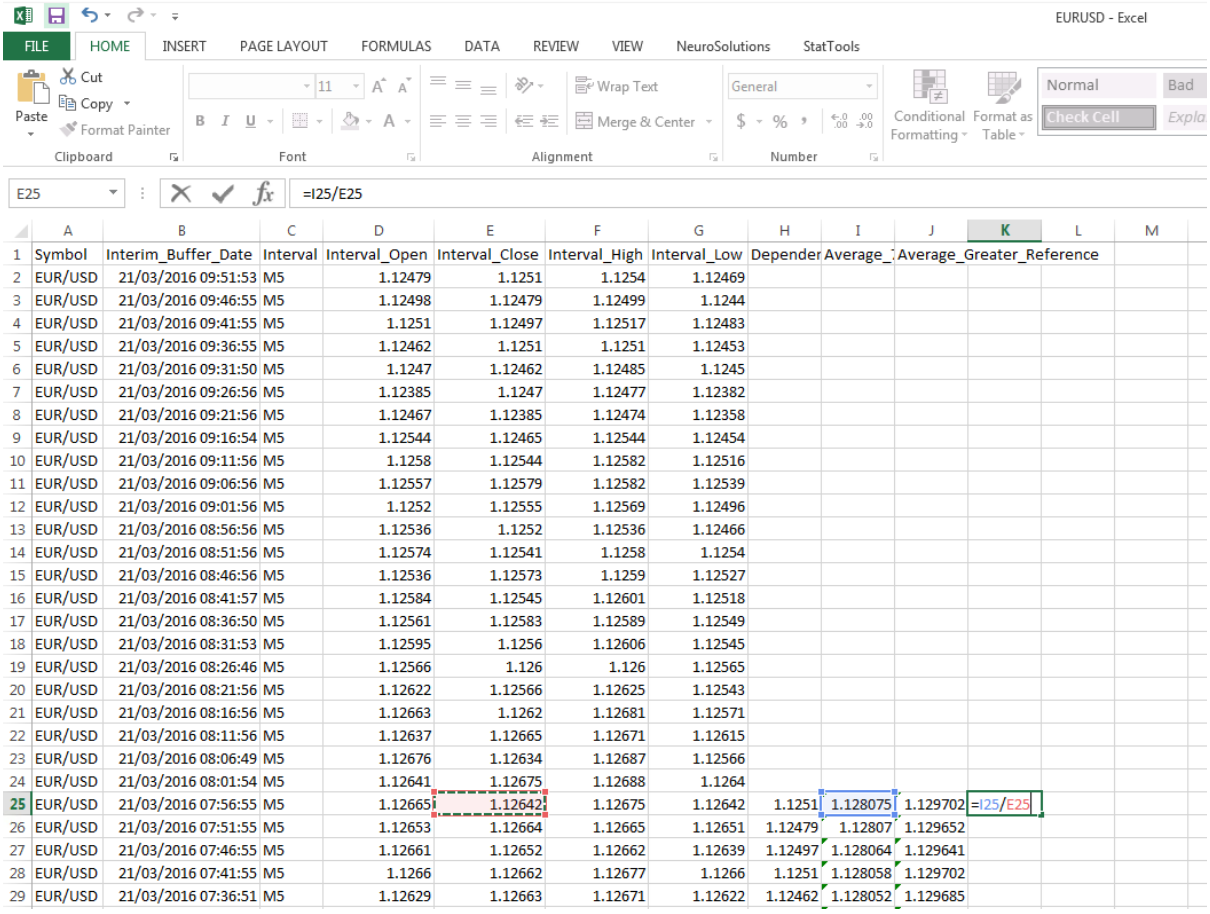
=I25/E25

E25 =I25/E25

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_700				
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469						
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244						
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483						
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453						
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245						
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382						
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358						
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454						
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516						
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539						
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496						
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466						
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254						
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527						
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518						
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549						
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545						
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565						
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543						
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571						
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615						
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566						
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264						
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	1.129702	=I25/E25		
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	1.129652			
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	1.129641			
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058	1.129702			
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052	1.129685			
30	EUR/USD	21/03/2016 07:31:54	M5	1.12626	1.1263	1.12636	1.12609	1.1247	1.128046	1.129595			
31	EUR/USD	21/03/2016 07:26:37	M5	1.12592	1.12626	1.12631	1.12591	1.12385	1.128042	1.129595			
32	EUR/USD	21/03/2016 07:21:56	M5	1.12646	1.12592	1.12646	1.12579	1.12465	1.128037	1.129612			
33	EUR/USD	21/03/2016 07:16:53	M5	1.1266	1.12646	1.12661	1.12631	1.12544	1.128032	1.129646			
34	EUR/USD	21/03/2016 07:11:55	M5	1.12655	1.12661	1.12664	1.12637	1.12579	1.128025	1.129652			
35	EUR/USD	21/03/2016 07:06:51	M5	1.1268	1.12655	1.12682	1.12637	1.12555	1.12802	1.129718			

The precedence of the Interval_Close is unimportant as long as it remains consistent throughput the abstraction and transformation.

Fill the variable down and name the column:



EURUSD - Excel

E25 =I25/E25

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_	Average_Greater_Reference			
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469						
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244						
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483						
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453						
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245						
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382						
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358						
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454						
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516						
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539						
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496						
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466						
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254						
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527						
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518						
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549						
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545						
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565						
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543						
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571						
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615						
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566						
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264						
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	1.129702	=I25/E25		
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	1.129652			
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	1.129641			
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058	1.129702			
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052	1.129685			

Although ratios, being the division of one Independent Variable against another, are an especially useful tool, the horizontal creation of Independent Variables may take advantage of a variety of arithmetic functions, where one value is being brought to bear against the next:

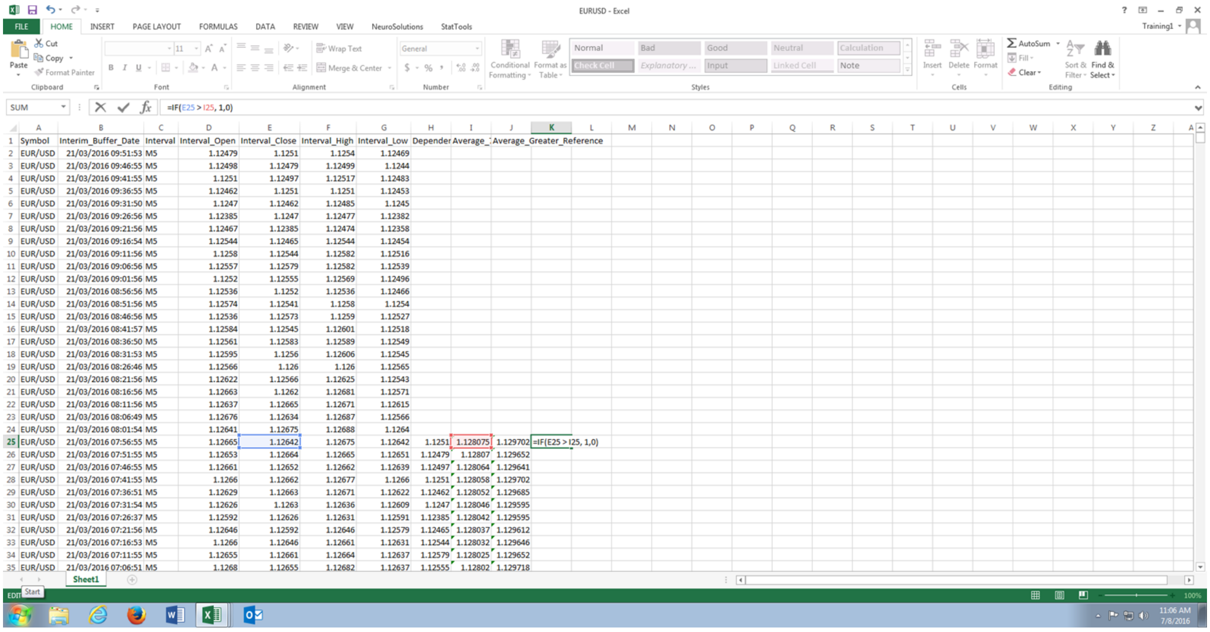
- Add: +
- Subtract: –
- Multiply: *

Procedure 5: Creating a Binary Independent Variable in Horizontal Abstraction.

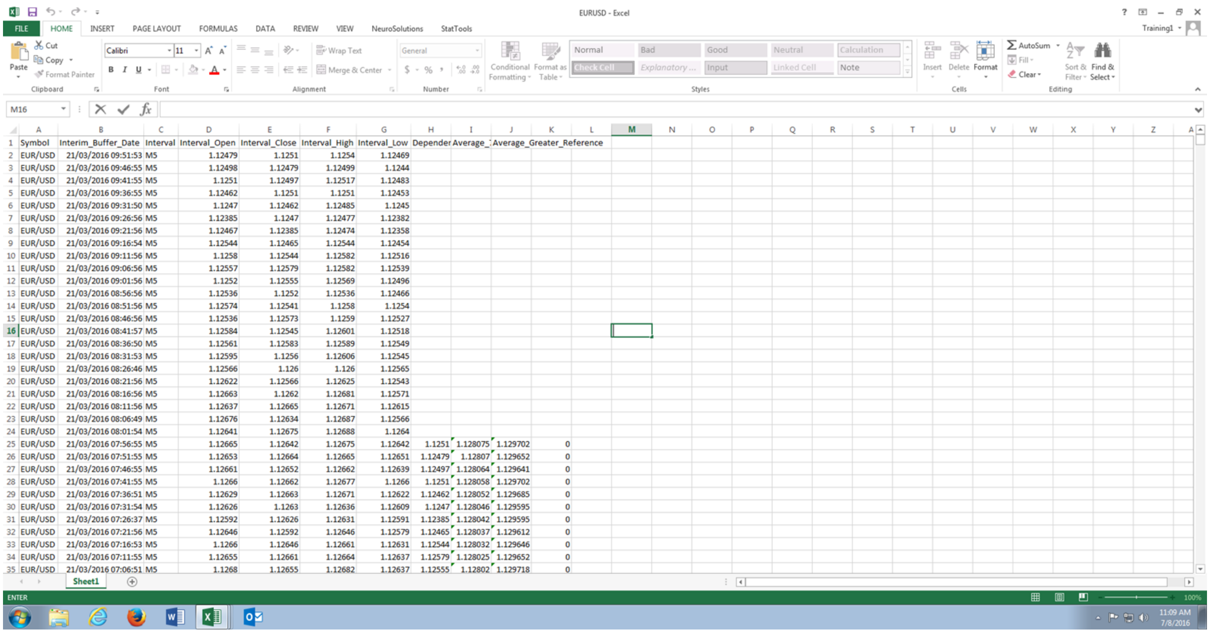
Binary Variables are an extremely good way to improve the performance of models where the data is not normally distributed, a maxim that is consistent across all modelling types introduced in this procedure guide. This procedure will use an IF function as a Horizontal Abstraction technique that will return 1 in the event that the prevailing Interval_Close is above the average as created in a Vertical Abstraction variable, in cell I25:

Follow the steps as set out in Procedure 11, instead using the following formula in the cell K25:

=IF(E25 > I25, 1,0)



Commit the formula, fill down and name the Independent Variable:



The IF function takes three parameters, the first being the condition followed the value to pivot to if true (i.e. 1) with the final parameter being the value to pivot in if false (i.e. 0). The IF function can use various conditional operators such as:

- Greater >
- Less <
- Greater Than or Equal >=
- Less Than or Equal <=
- Not Equal <>

The IF statement should be used creatively across several Independent Variable combinations and operator types as part of a creative Abstraction process.

JUBE

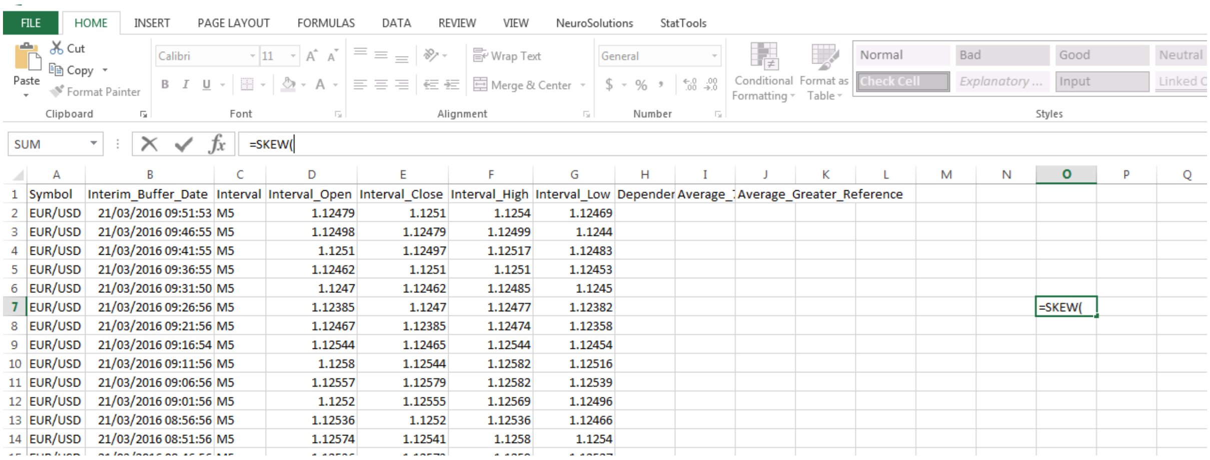
Procedure 6: Creating a Statistical Transformation using SQRT and observing improvement.

In the same manner as Procedure 12 uses the IF function for the purposes of Horizontal Abstraction, there are a plethora of other functions that can provide statistical transformation, which is most generally used to correct data where abnormally distributed (i.e. not a normal distribution), with a view to the independent variable becoming more normally distributed.

Firstly, it is necessary to understand the overall need for a statistical abstraction, such as SQRT or LOG. To gain a very quick measure of the direction of lean the Skew function can be used, although this is only one or a number of measures to be considered when appraising distribution properties.

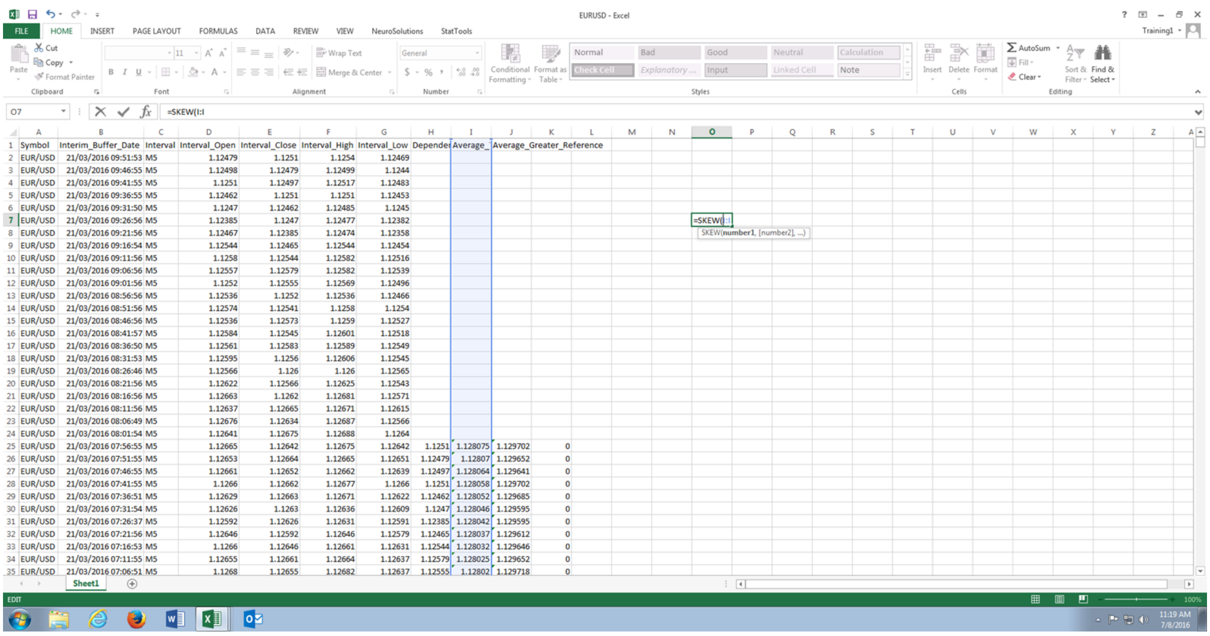
Click on a free cell, in this example O7, then begin typing the formula:

=SKEW(

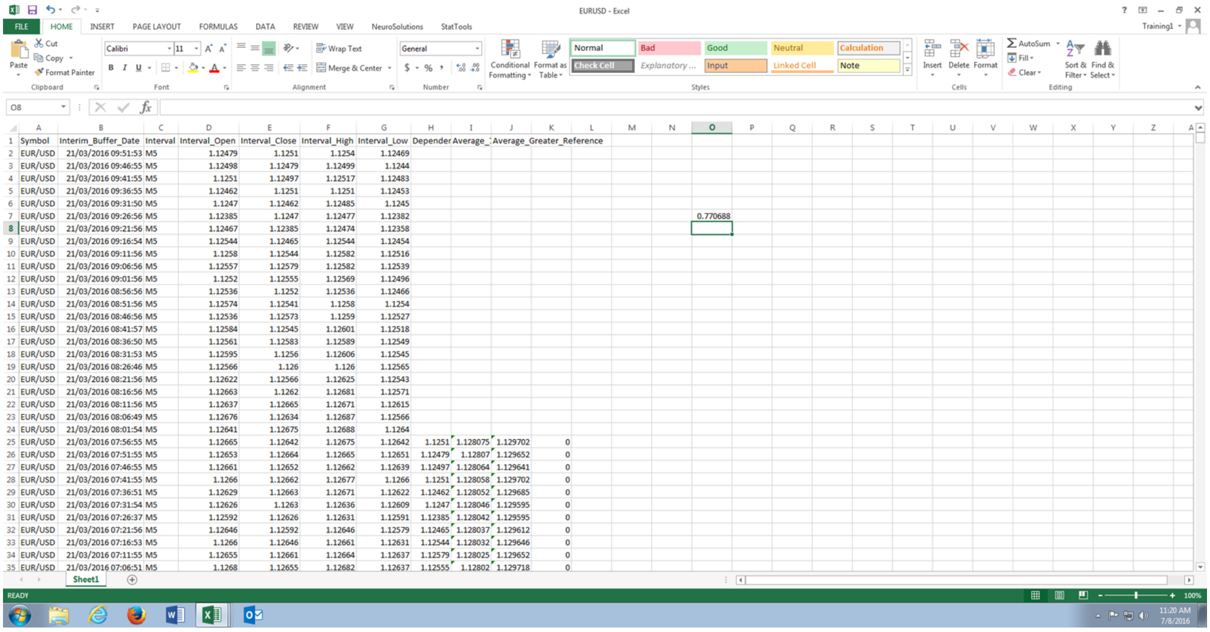


Click on the column containing the vertical abstraction independent variable Average_700, which could also be expressed as I:I:

=SKEW(I:I

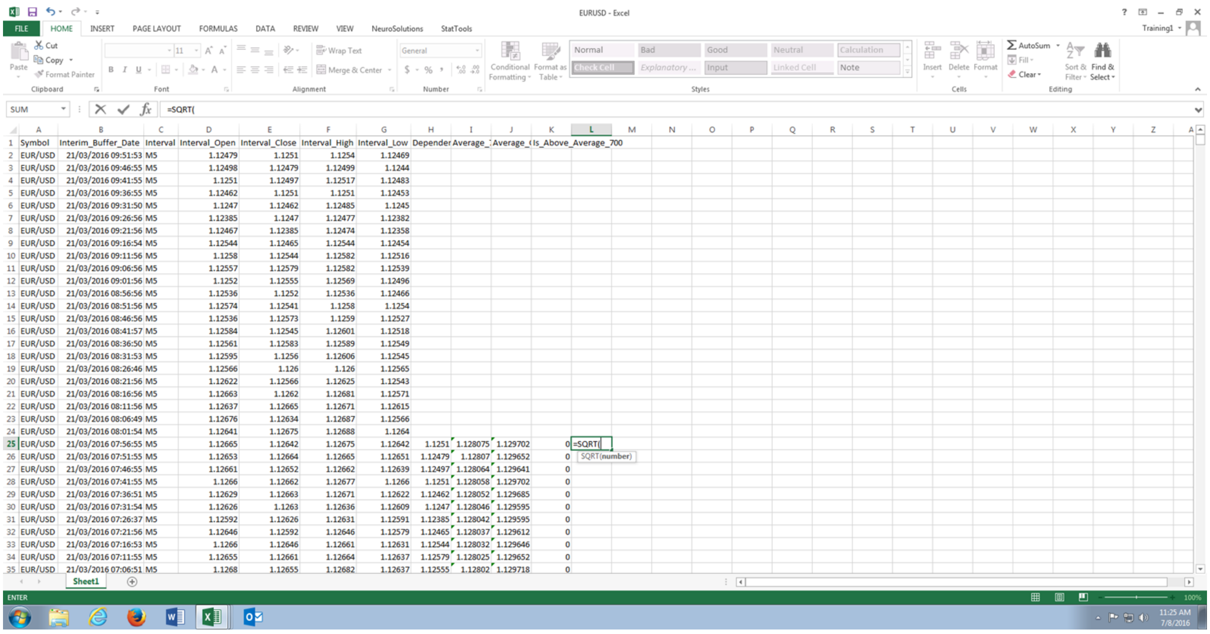


Complete the formula by including the closing parenthesis. In this example observe the SKEW to be slightly positive, leaning towards the axis, conceptually:



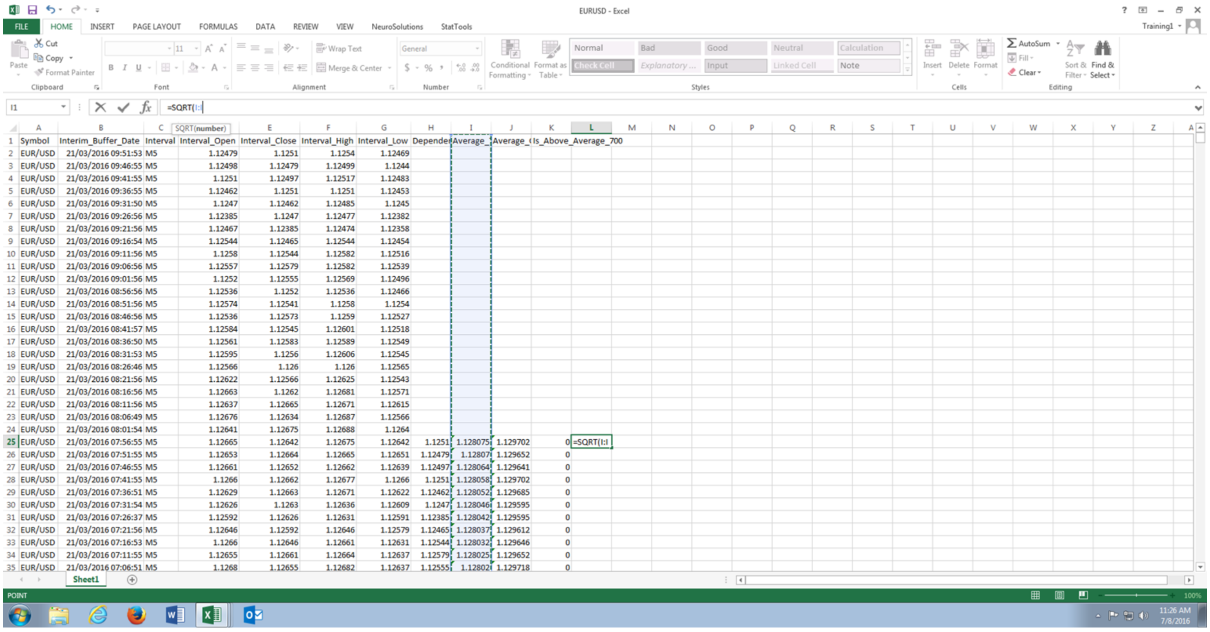
The purpose of this procedure is to observe the reduction in this skew by using the SQRT statistical transformation. Follow procedure 11 as if to create an IF horizontal abstraction, while typing the beginning of formula:

= SQRT(

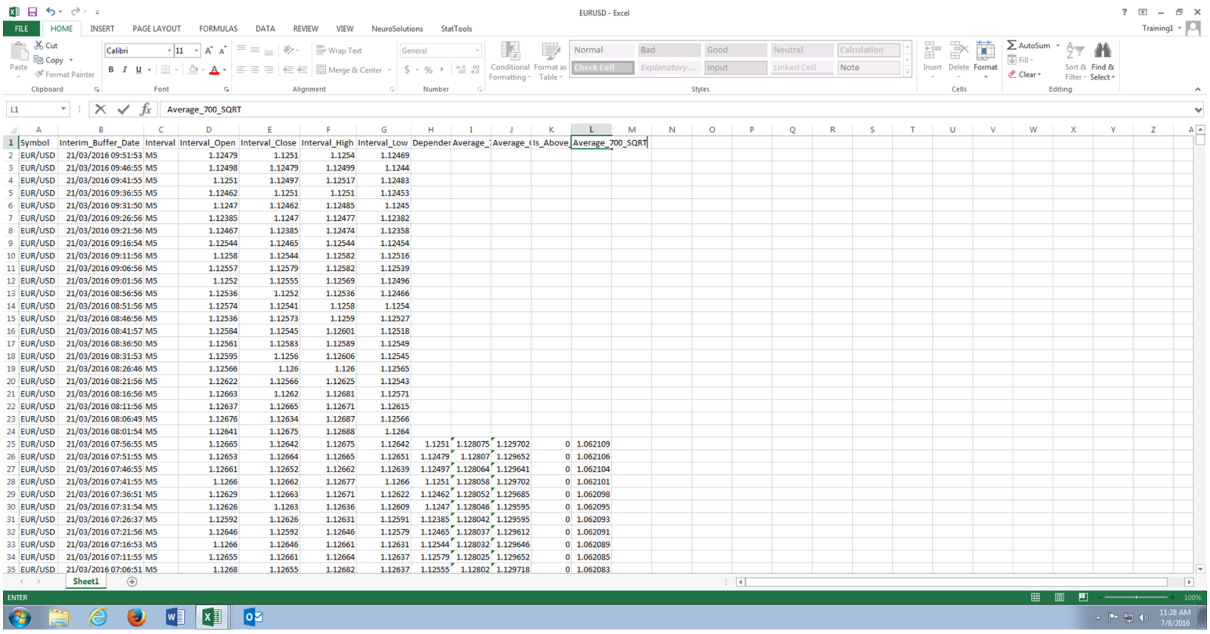


Click on the column titled Average_700, which in this case could also be expressed as I:I:

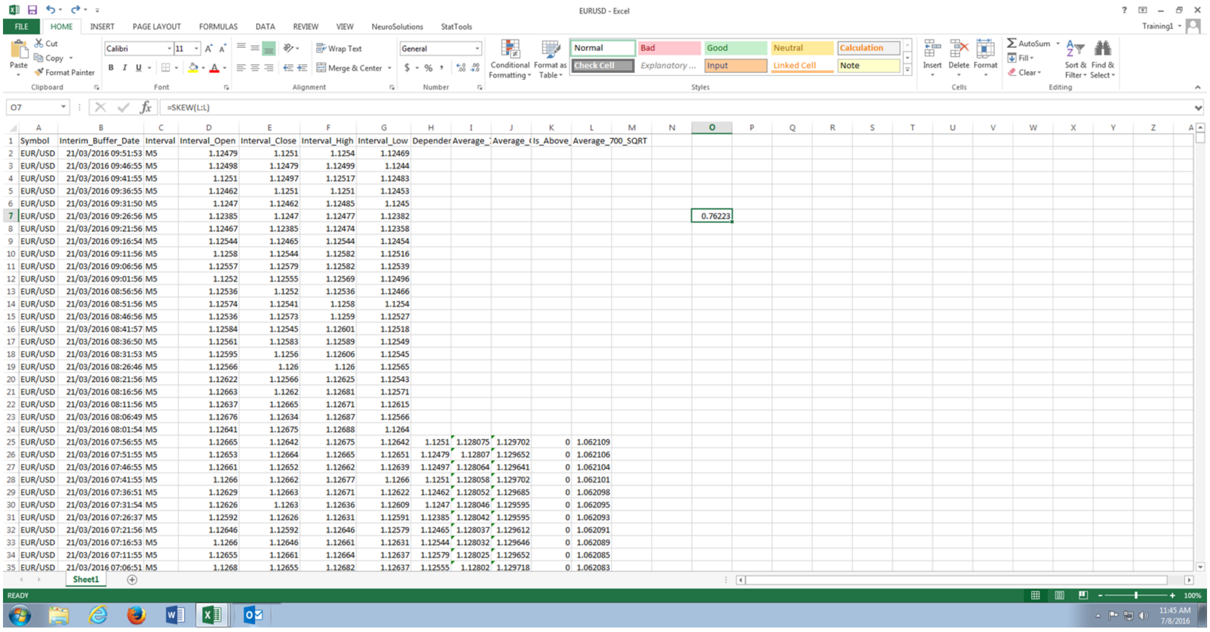
=SQRT(I:I



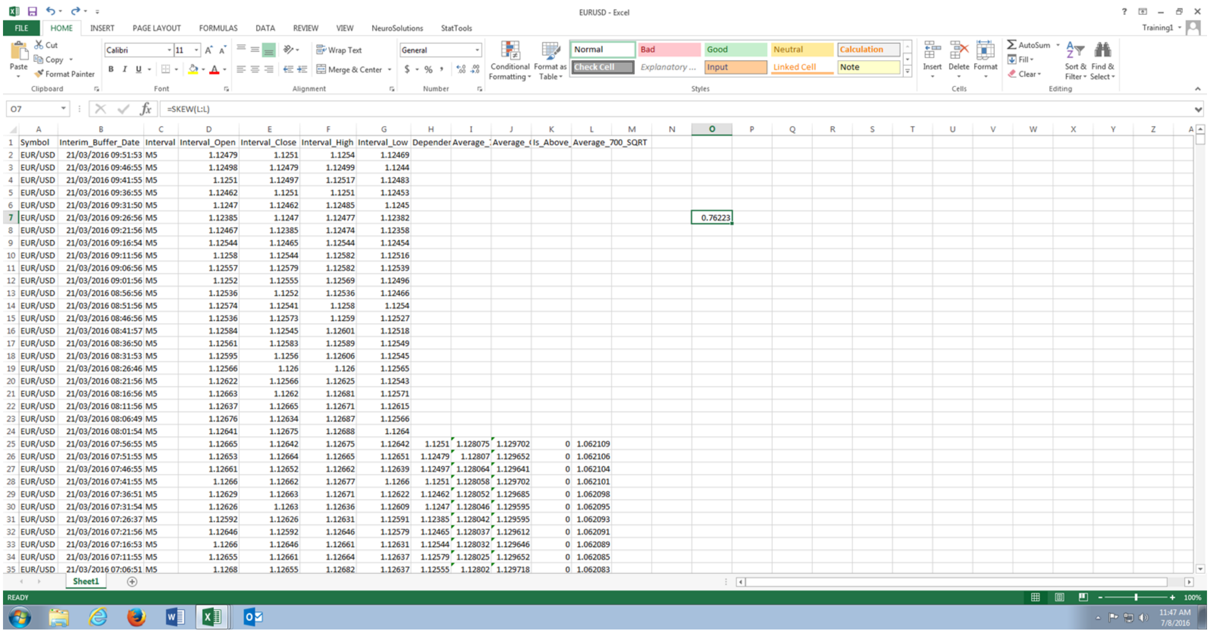
Close the parenthesis to complete the formula. Fill down and name the column Average_700_SQRT:



Repeat this procedure to identify the SKEW of the new abstracted independent variable instead of I:I, use J:J:



It can be observed that a very modest improvement in the SKEW has been observed, which appears to be quite underwhelming:



This transformation can have a more profound effect however when carried forward to the modelling techniques that follow in this guide.

Other transformations can be performed in the same manner such as:

- Logarithm: LOG
- Absolute: ABS
- Power: POWER

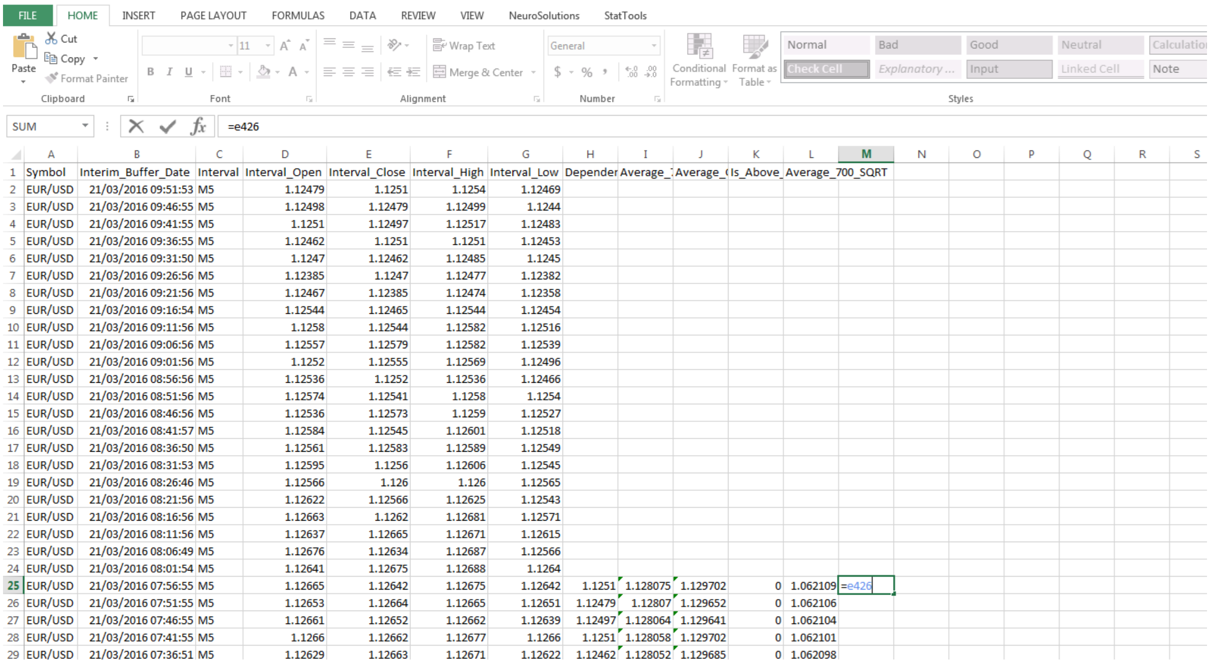
While it is certainly advisable to attempt such statistical transformations, given time enough in the abstraction process, it is generally recommended to perform more work on abstracting independent variables in an effort to compensate for data which is not normally distributed.

Procedure 7: Creating a Point Independent Variable in time series data.

Point Independent Variables can be used as a more explicit means to identify how a preceding value, although quite often an index or additional value, may have some leading indicative effect on the current value. Given a specific scope, point variables should be selected at evenly increasing points in that scope, for example, Point 700, Point 600 etc.

In this example, out of 700 possible intervals available in scope, we are going to select the price at point 300. Execute procedure 11, instead entering the formula targeting the point 300 intervals into the scope (bottom up):

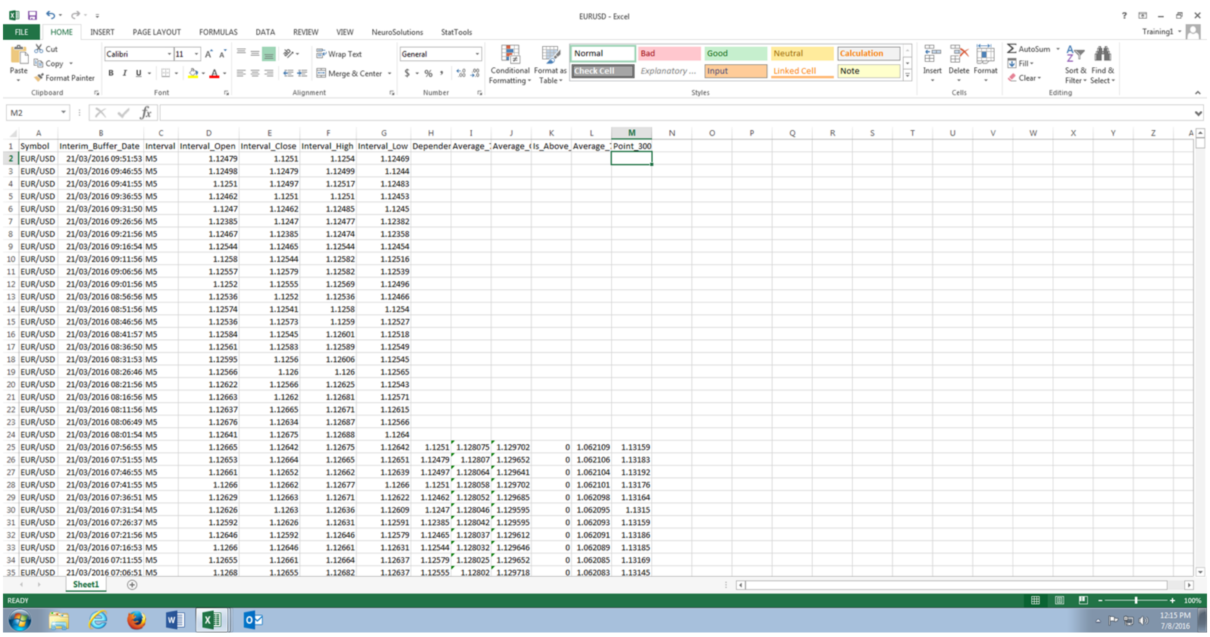
=E426



SUM =e426

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_	Average_(	Is_Above_	Average_700_SQRT	
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469						
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244						
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483						
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453						
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245						
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382						
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358						
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454						
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516						
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539						
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496						
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466						
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254						
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527						
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518						
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549						
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545						
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565						
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543						
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571						
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615						
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566						
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264						
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	1.129702	0	1.062109	=e426
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	1.129652	0	1.062106	
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	1.129641	0	1.062104	
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058	1.129702	0	1.062101	
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052	1.129685	0	1.062098	

Simply commit the formula, fill down and name the column Point_300.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depender	Average_	Average_(	Is_Above_	Average_	Point_300
2	EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469						
3	EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244						
4	EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483						
5	EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453						
6	EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245						
7	EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382						
8	EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358						
9	EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454						
10	EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516						
11	EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539						
12	EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496						
13	EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466						
14	EUR/USD	21/03/2016 08:51:56	M5	1.12574	1.12541	1.1258	1.1254						
15	EUR/USD	21/03/2016 08:46:56	M5	1.12536	1.12573	1.1259	1.12527						
16	EUR/USD	21/03/2016 08:41:57	M5	1.12584	1.12545	1.12601	1.12518						
17	EUR/USD	21/03/2016 08:36:50	M5	1.12561	1.12583	1.12589	1.12549						
18	EUR/USD	21/03/2016 08:31:53	M5	1.12595	1.1256	1.12606	1.12545						
19	EUR/USD	21/03/2016 08:26:46	M5	1.12566	1.126	1.126	1.12565						
20	EUR/USD	21/03/2016 08:21:56	M5	1.12622	1.12566	1.12625	1.12543						
21	EUR/USD	21/03/2016 08:16:56	M5	1.12663	1.1262	1.12681	1.12571						
22	EUR/USD	21/03/2016 08:11:56	M5	1.12637	1.12665	1.12671	1.12615						
23	EUR/USD	21/03/2016 08:06:49	M5	1.12676	1.12634	1.12687	1.12566						
24	EUR/USD	21/03/2016 08:01:54	M5	1.12641	1.12675	1.12688	1.1264						
25	EUR/USD	21/03/2016 07:56:55	M5	1.12665	1.12642	1.12675	1.12642	1.1251	1.128075	1.129702	0	1.062109	1.13159
26	EUR/USD	21/03/2016 07:51:55	M5	1.12653	1.12664	1.12665	1.12651	1.12479	1.12807	1.129652	0	1.062106	1.13183
27	EUR/USD	21/03/2016 07:46:55	M5	1.12661	1.12652	1.12662	1.12639	1.12497	1.128064	1.129641	0	1.062104	1.13192
28	EUR/USD	21/03/2016 07:41:55	M5	1.1266	1.12662	1.12677	1.1266	1.1251	1.128058	1.129702	0	1.062101	1.13176
29	EUR/USD	21/03/2016 07:36:51	M5	1.12629	1.12663	1.12671	1.12622	1.12462	1.128052	1.129685	0	1.062098	1.13164
30	EUR/USD	21/03/2016 07:31:54	M5	1.12626	1.1263	1.12636	1.12609	1.1247	1.128046	1.129595	0	1.062095	1.1315
31	EUR/USD	21/03/2016 07:26:37	M5	1.12592	1.12626	1.12631	1.12591	1.12385	1.128042	1.129595	0	1.062093	1.13159
32	EUR/USD	21/03/2016 07:21:56	M5	1.12646	1.12592	1.12646	1.12579	1.12465	1.128037	1.129612	0	1.062091	1.13186
33	EUR/USD	21/03/2016 07:16:53	M5	1.1266	1.12646	1.12661	1.12631	1.12544	1.128032	1.129646	0	1.062089	1.13185
34	EUR/USD	21/03/2016 07:11:55	M5	1.12655	1.12661	1.12664	1.12637	1.12579	1.128025	1.129652	0	1.062085	1.13169
35	EUR/USD	21/03/2016 07:06:51	M5	1.1268	1.12655	1.12682	1.12637	1.12555	1.12802	1.129718	0	1.062083	1.13145

Repeat the steps to create even point references around this example (for example, Point_100, Point_200, Point_300) based on a preference for even spacing between points.

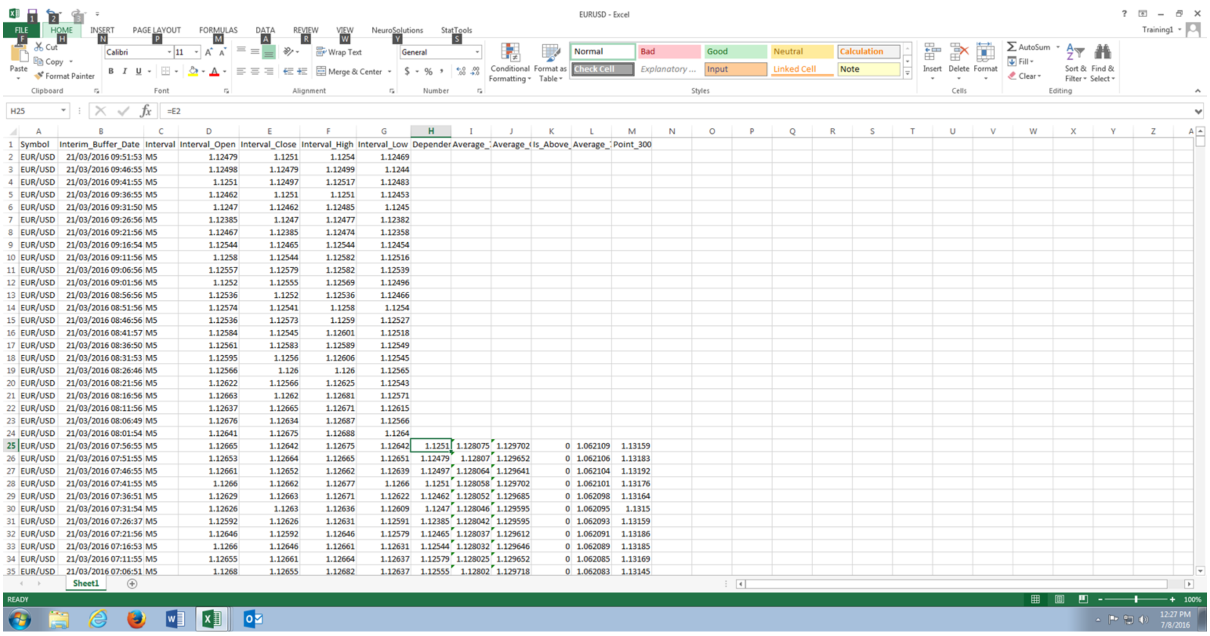
Procedure 8: Anchor an Independent Variable and a Dependent Variable.

When performing time series analysis, thereafter predictive modelling, it is generally not advisable to seek to predict a raw value in a horizon, rather seek to predict the change in that value in a horizon. Predicting a raw value, unless the variables obey strict boundaries, will typically exhibit in that a model performs exceptionally well in test, yet not that well at all in production. It follows that a variable must be anchored to a variable representing the current \ prevailing environment, which in our example is restricted to a single variable of Interval_Close in the example to be predicted.

For each variable created in abstraction, this would include both dependent and independent variables, repeating the process as follows. In our example we will anchor the dependent variable.

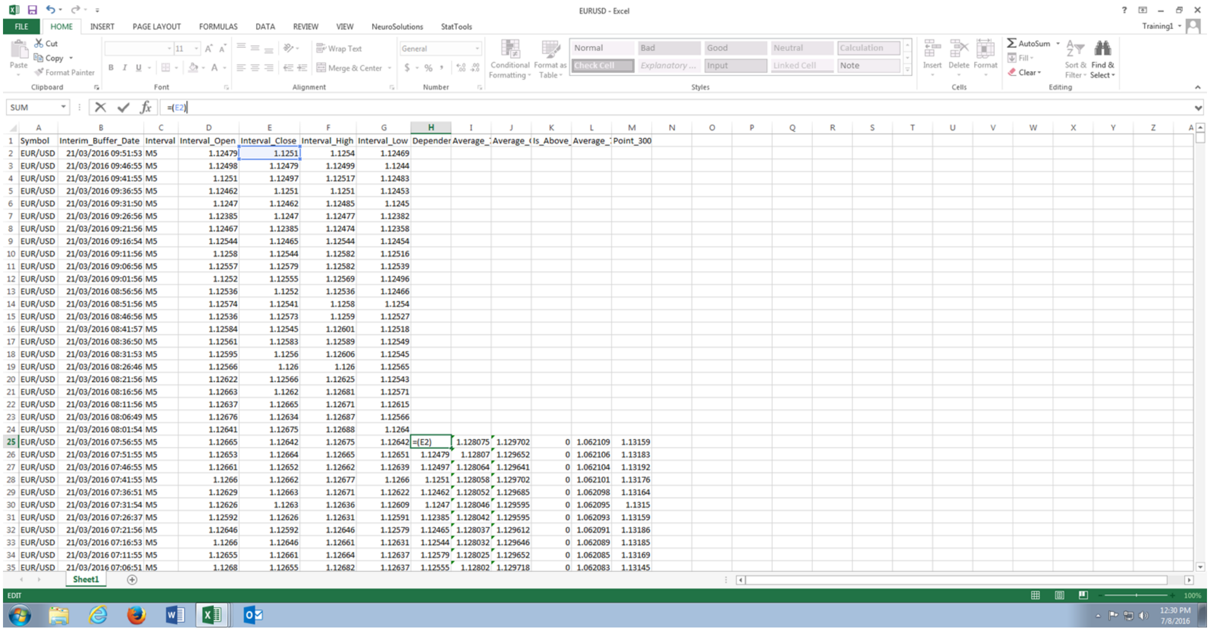
Select the cell containing the dependent variable calculation, in our example this is H25, referencing price in the intervals ahead by two hours (i.e 24 examples):

=E2



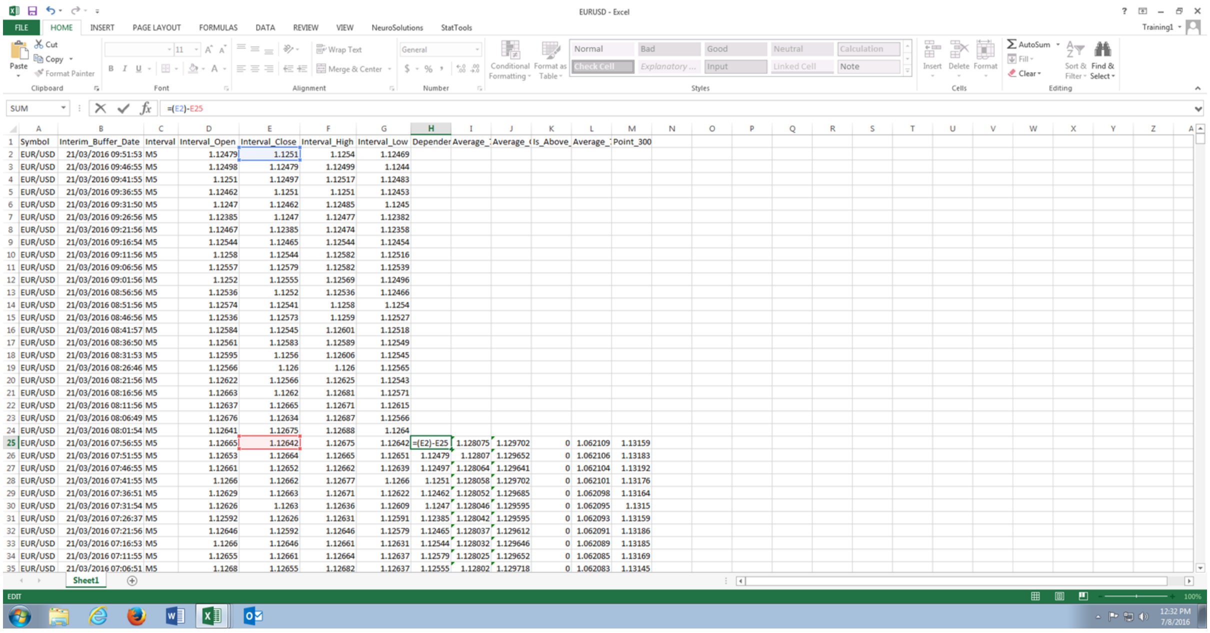
As a point of protocol, wrap the existing formula in parenthesis to effortlessly manage order of precedence in more complex variables:

=(E2)



To anchor the dependent variable to reflect change, simply subtract the prevailing price contained in cell E25. It follows that the change, rather than the raw value, is now reflected in the formula:

=(E2)-E25

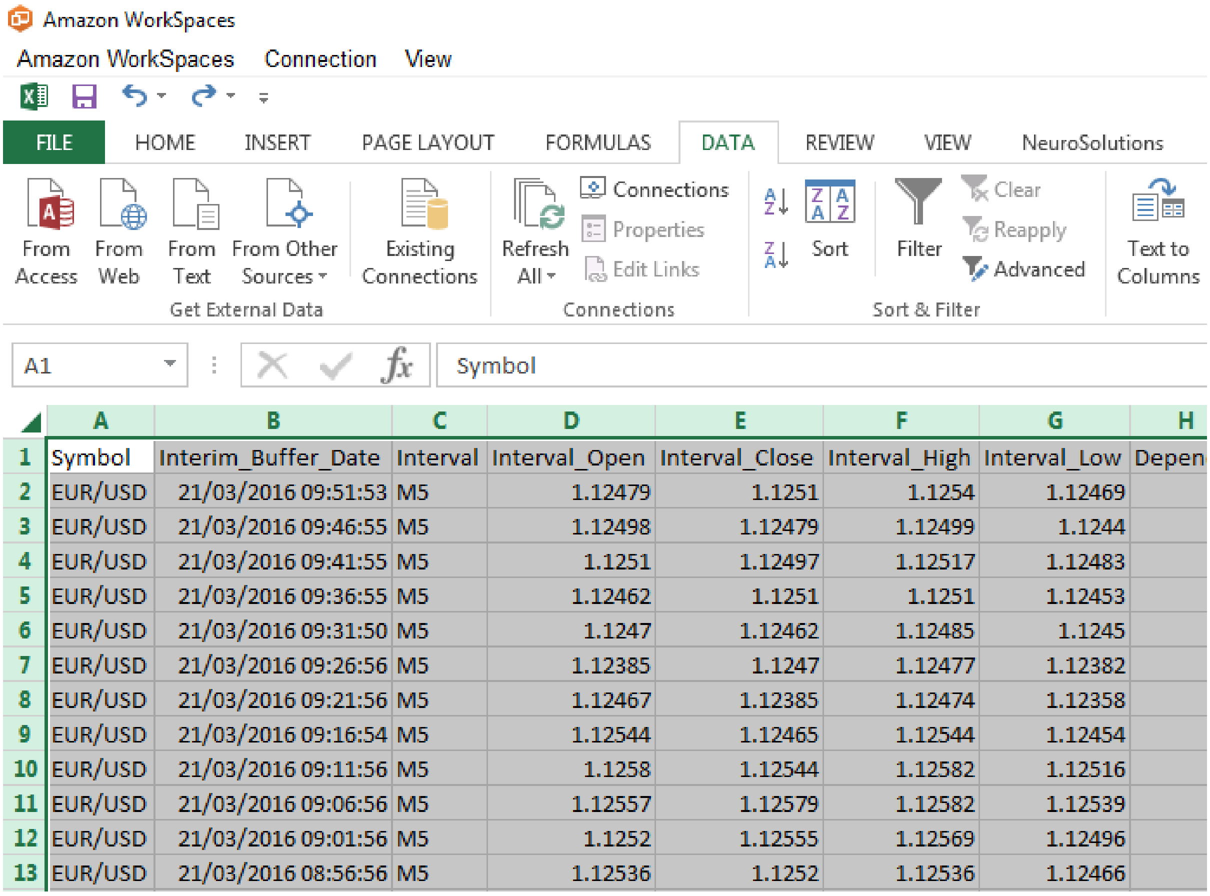


Commit the formula, fill down and repeat for each variable so that all independent variables are anchored in the same manner as the dependent variable.

Procedure 9: Clean Up, Remove Formulas and Save Abstraction File.

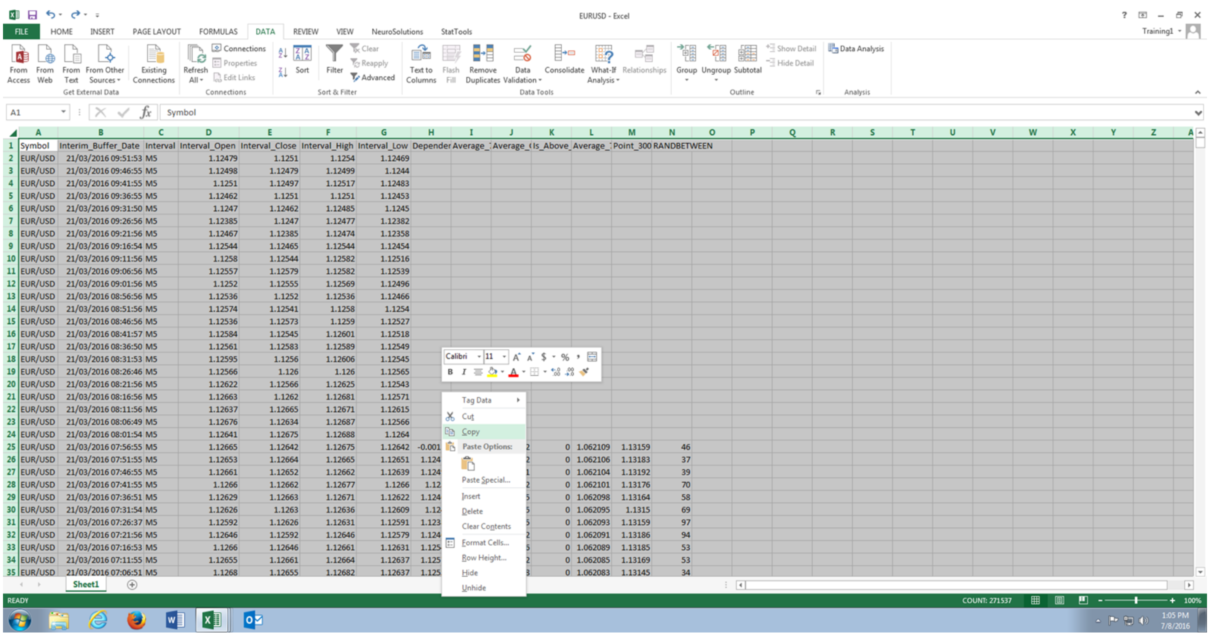
Having created a spreadsheet of numerous abstracted independent variables, the file must be finalised for the purposes of sampling and model creation. This finalisation of a file would involve removing uncompleted intervals (i.e. the top 24 intervals in the spreadsheet) and intervals which do not have at a minimum the required amount of intervals in scope (i.e. the bottom 700 intervals).

Firstly, remove all formulas. Selecting the entire spreadsheet by selecting in the top left hand corner of the workbook:

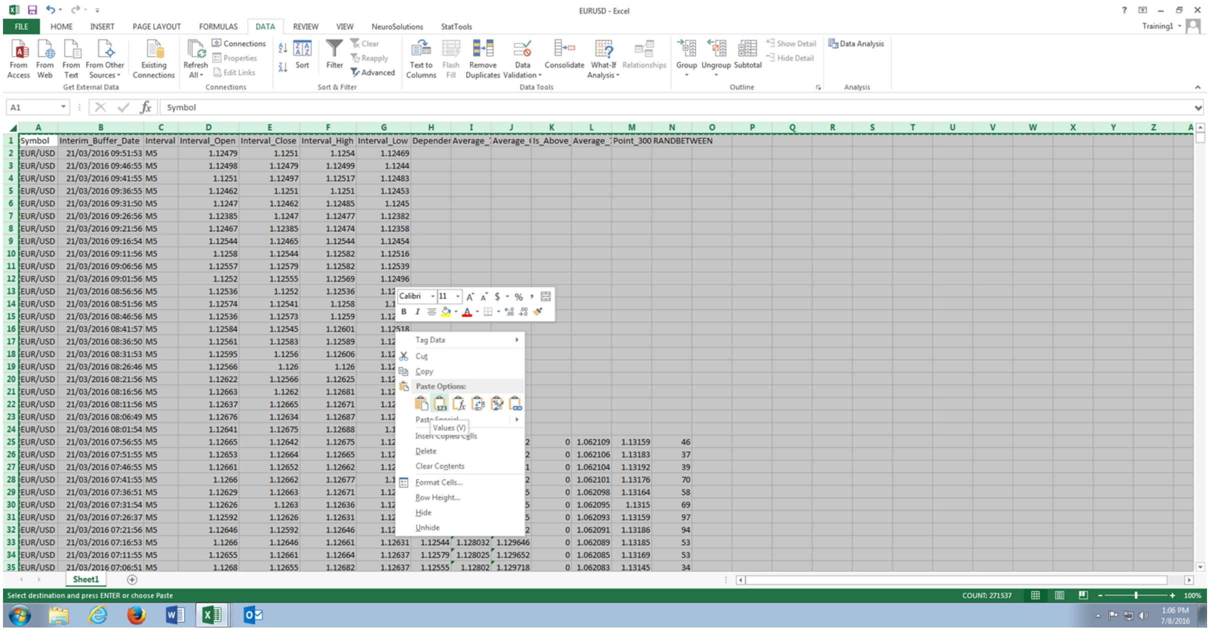


Symbol	Interim_Buffer_Date	Interval	Interval_Open	Interval_Close	Interval_High	Interval_Low	Depen
EUR/USD	21/03/2016 09:51:53	M5	1.12479	1.1251	1.1254	1.12469	
EUR/USD	21/03/2016 09:46:55	M5	1.12498	1.12479	1.12499	1.1244	
EUR/USD	21/03/2016 09:41:55	M5	1.1251	1.12497	1.12517	1.12483	
EUR/USD	21/03/2016 09:36:55	M5	1.12462	1.1251	1.1251	1.12453	
EUR/USD	21/03/2016 09:31:50	M5	1.1247	1.12462	1.12485	1.1245	
EUR/USD	21/03/2016 09:26:56	M5	1.12385	1.1247	1.12477	1.12382	
EUR/USD	21/03/2016 09:21:56	M5	1.12467	1.12385	1.12474	1.12358	
EUR/USD	21/03/2016 09:16:54	M5	1.12544	1.12465	1.12544	1.12454	
EUR/USD	21/03/2016 09:11:56	M5	1.1258	1.12544	1.12582	1.12516	
EUR/USD	21/03/2016 09:06:56	M5	1.12557	1.12579	1.12582	1.12539	
EUR/USD	21/03/2016 09:01:56	M5	1.1252	1.12555	1.12569	1.12496	
EUR/USD	21/03/2016 08:56:56	M5	1.12536	1.1252	1.12536	1.12466	

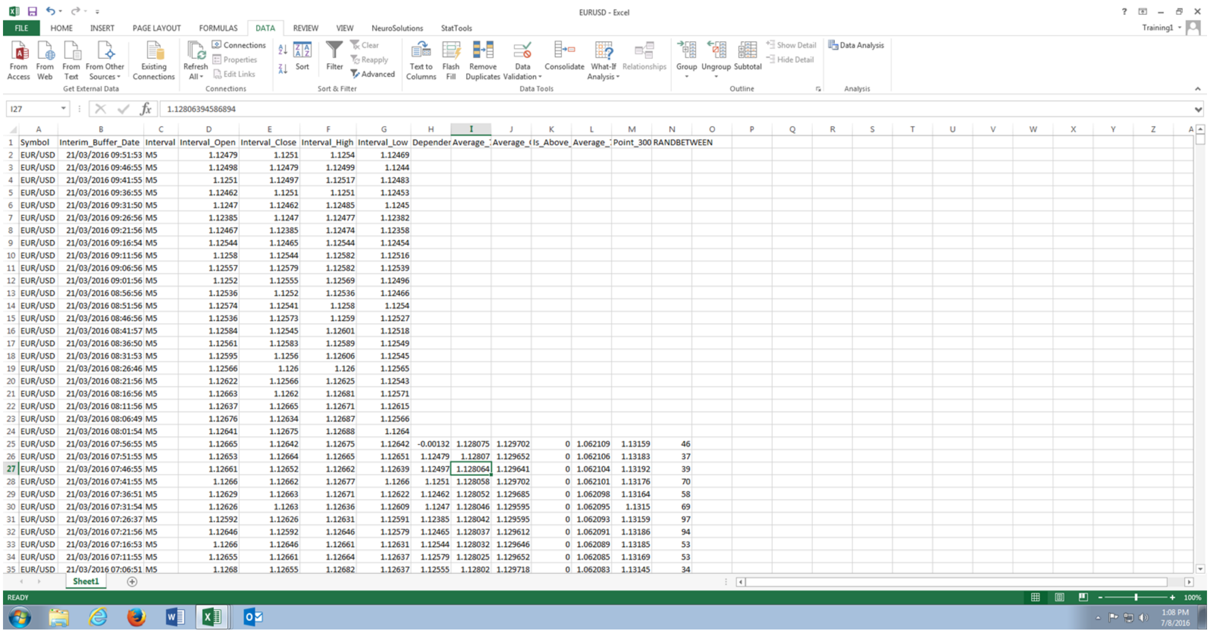
Right click and select copy:



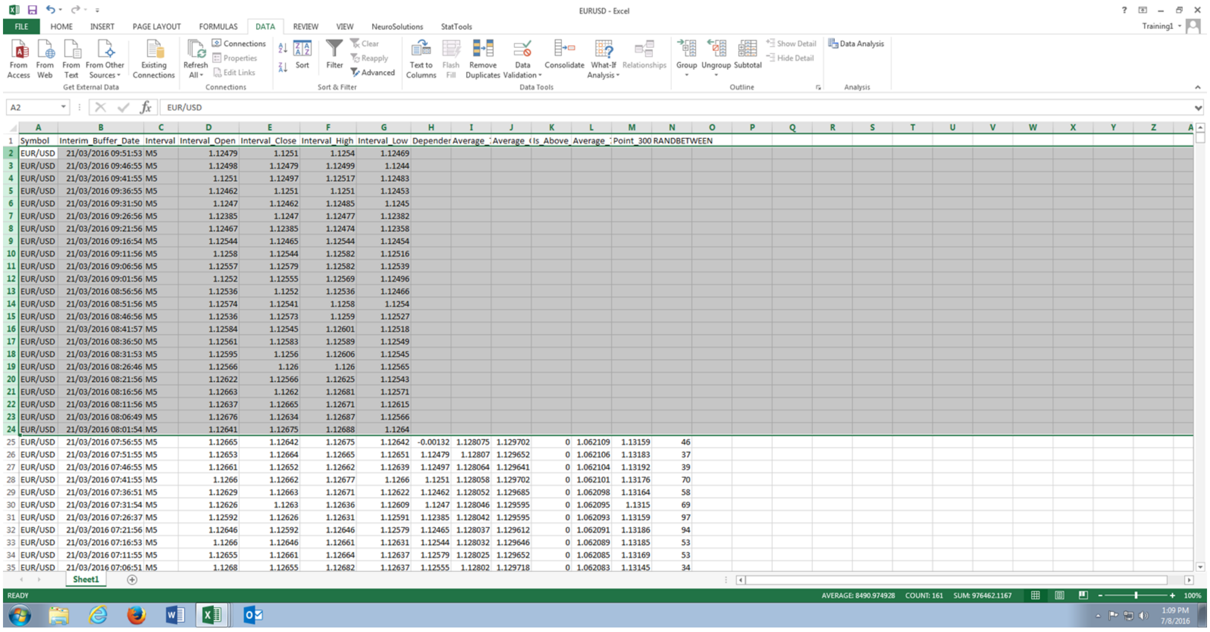
Right click and under Paste Options, select Values:



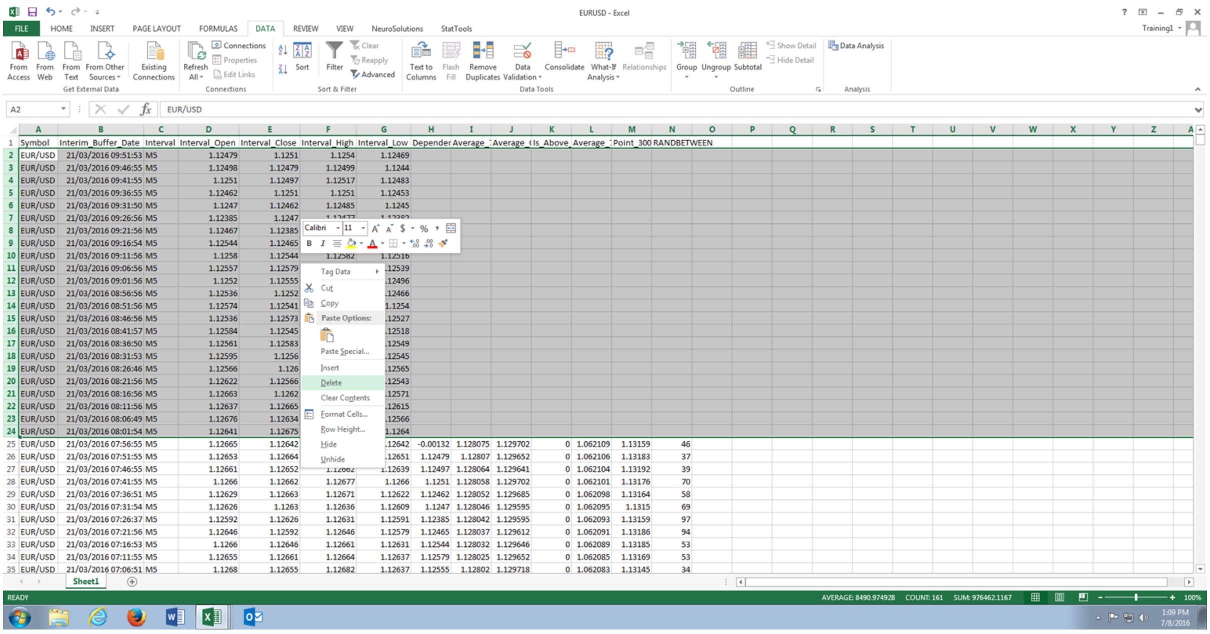
This will execute a copy, then paste over whereby only the cell values are included, thus removing the formulas. As such, when examples are deleted there will be no effect on the cell values:



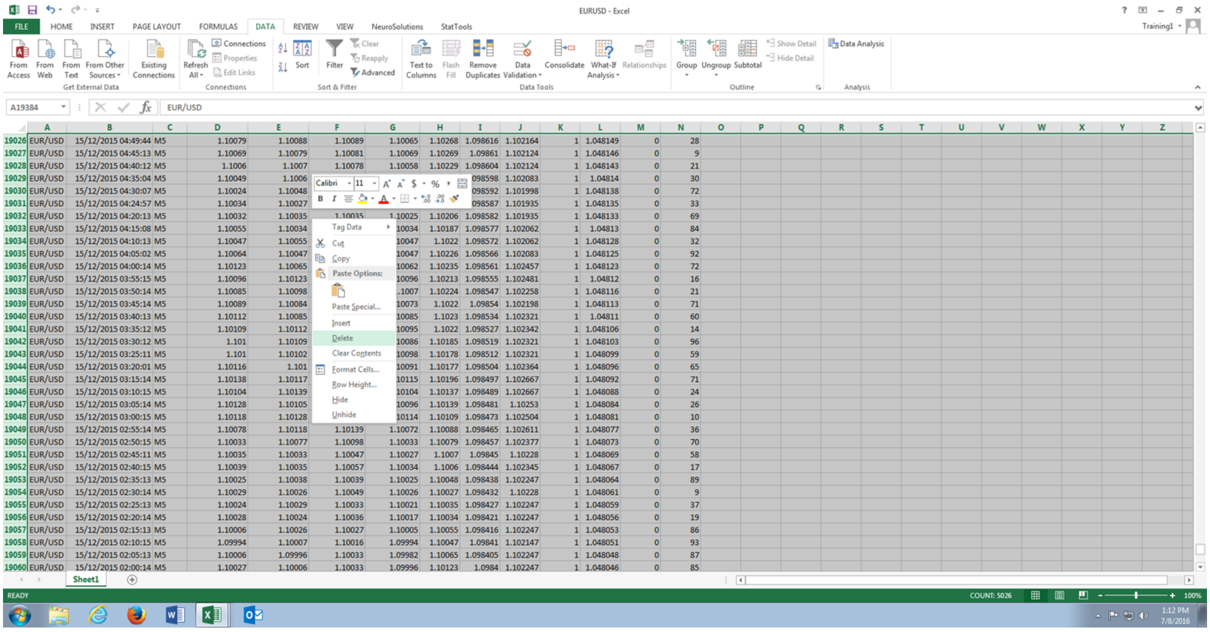
Select the top 24, being the horizon uncompleted intervals examples in the workbook:



Right click, and click delete:



Select the bottom 700 examples in the workbook, right click and click delete:



The file is now ready for predictive analytics modelling in any of the techniques as follows.

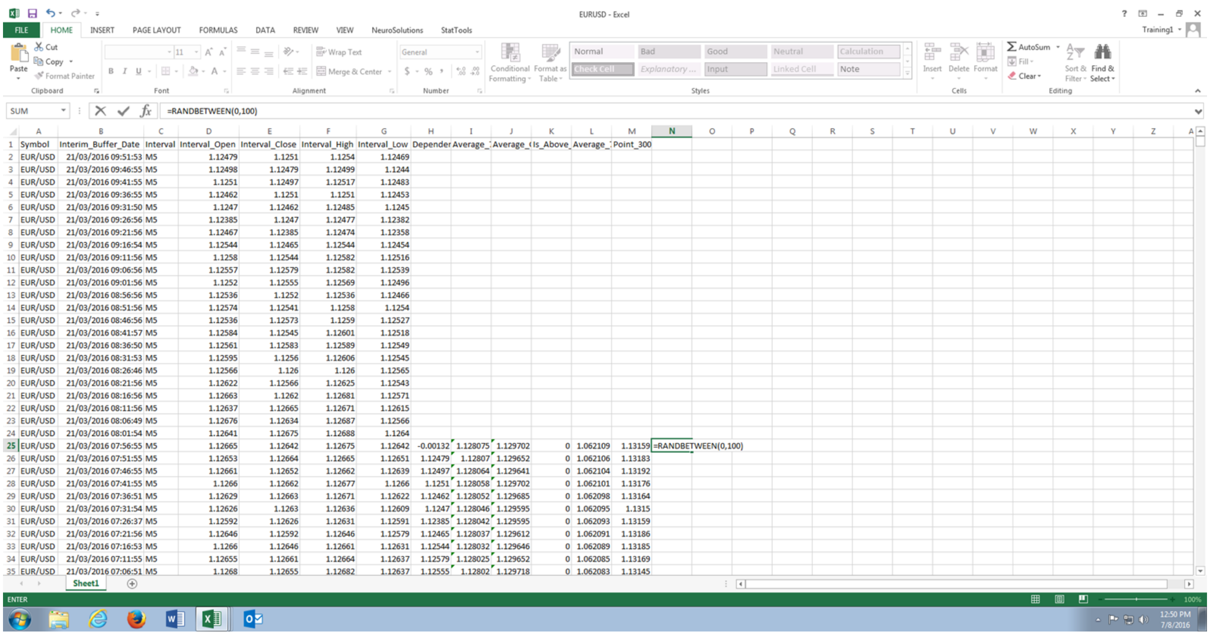
Procedure 10: Assign a Random Digit for Sampling.

Before attempting this procedure, ensure that the abstraction file has been cleaned and finalised (i.e. there are no formulas remaining) as per procedure 16.

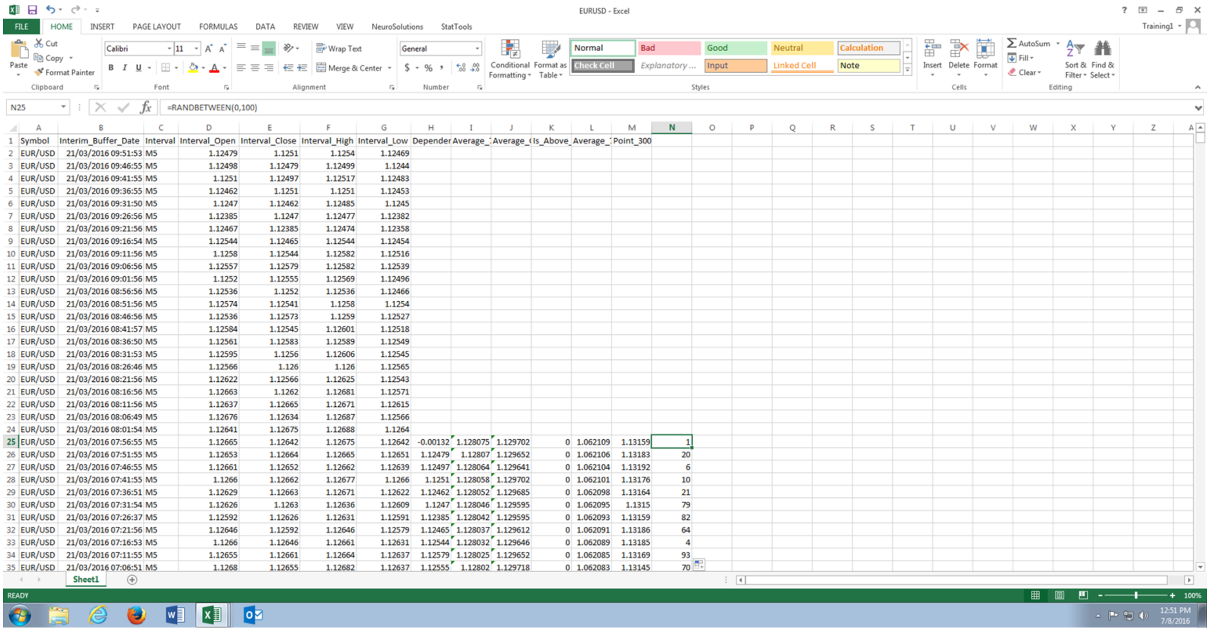
The file being used in this example has some 20k examples, which is manageable but perhaps a little overwhelming for ad-hoc, summary statistics led exploratory analysis. While sampling is not absolutely mandatory at this volume of data, the principle can be applied to much larger datasets.

Execute Procedure 11, instead implementing a RANDBETWEEN function to create a random digit between 0 and 100:

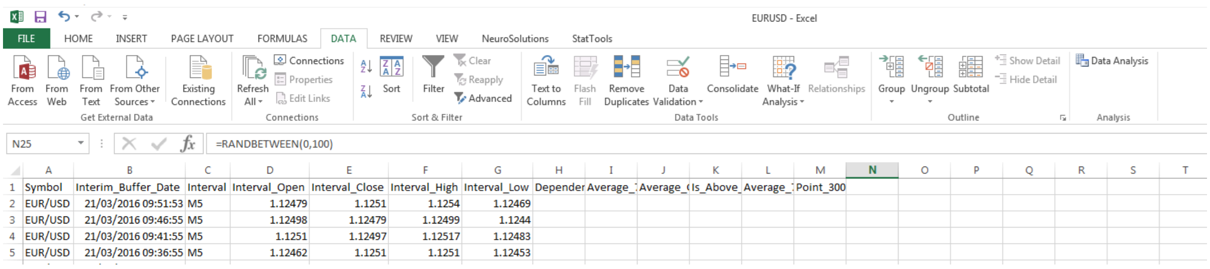
=RANDBETWEEN(0,100)



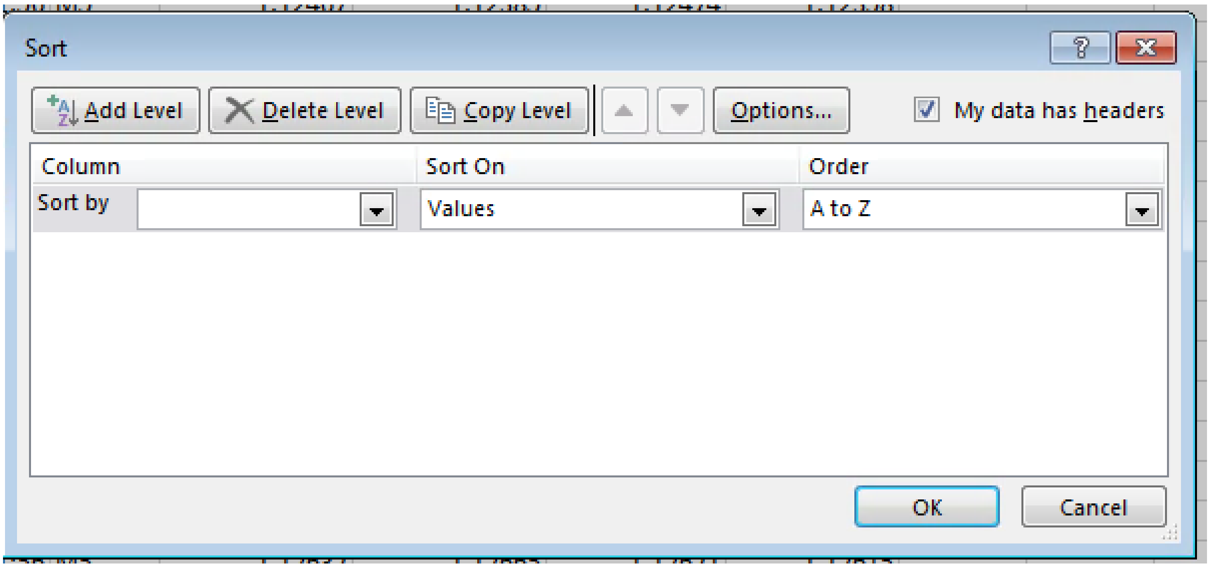
Commit the formula, fill down and name the column Random Digit:



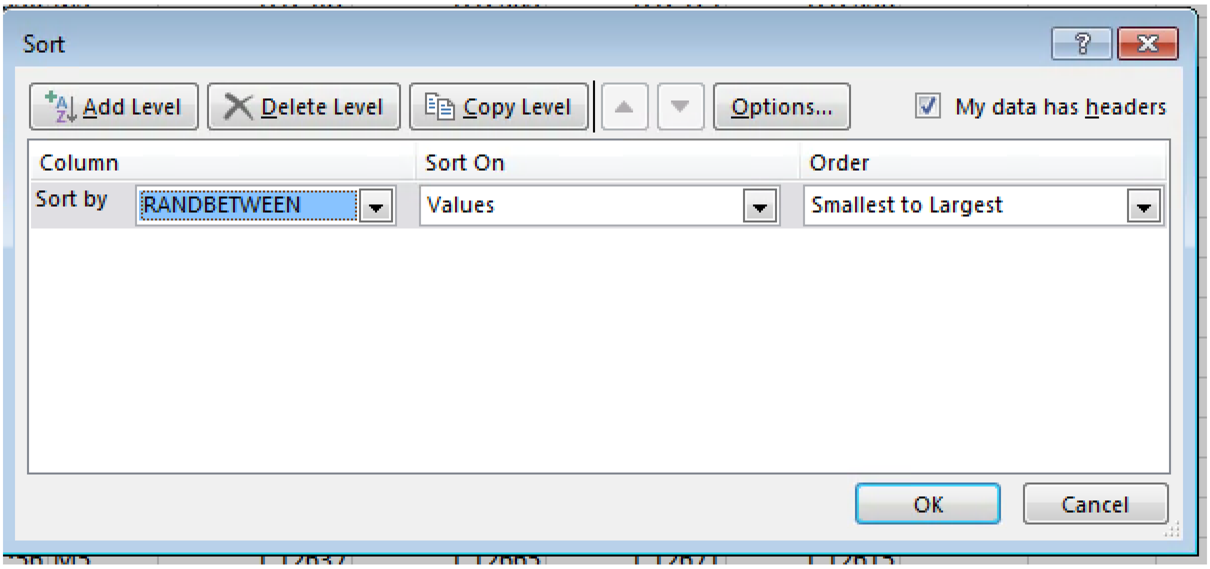
Select the Data Ribbon:



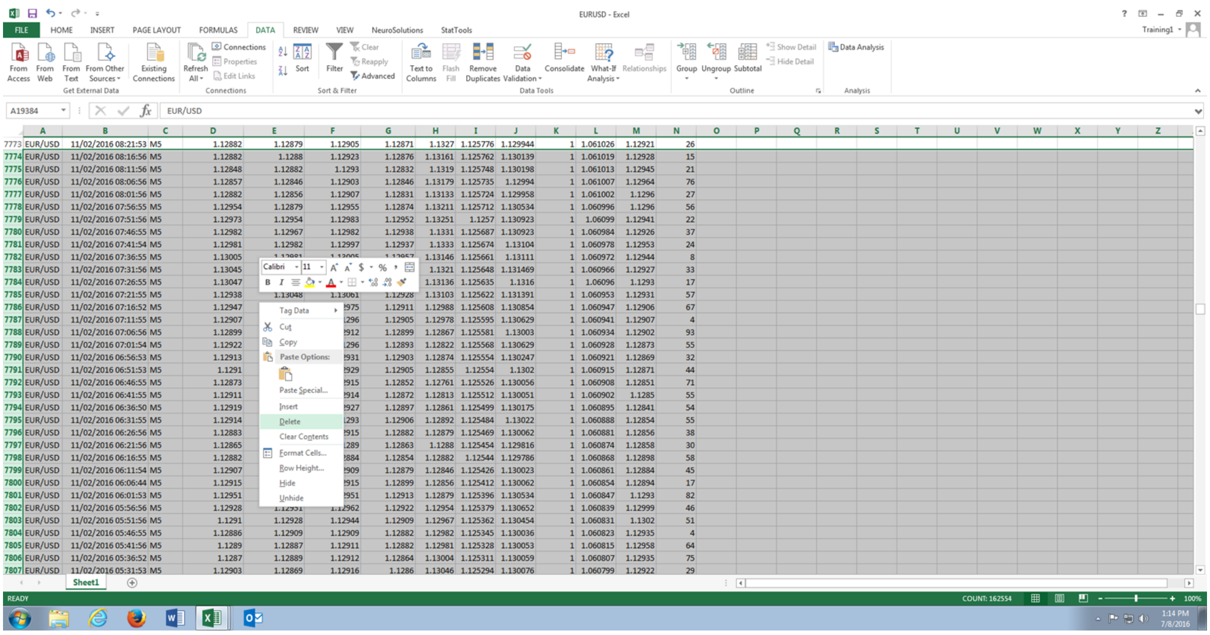
Select the Icon Sort, towards the centre of the ribbon which will open the sort window:



The column to sort by is the newly created RANDBETWEEN Independent Variable, of which the Order is unimportant. Select Sort By RANDBETWEEN, then click OK:



The dataset will now be in Random Order and as such records can be removed until the dataset is of a size deemed manageable for analysis. In or example, scrolling to the bottom or the dataset and selecting the bottom 10000 records, right clicking and clicking delete will create a more manageable dataset, while being just as statistically meaningful:



Module 7: ggplot2 Rapid Exploration

Up to this point in the procedures the plot() and hist() function has been used, which invokes the base graphics function of the R software. It can't be said that base R graphics have the aesthetic properties of charts produced by rivals such as Excel and leaves a lot to be desired for the purposes of presentation. Fortunately, there is a more powerful package that is available in R for producing stunning charts that will be at home in any presentation, ggplot2.

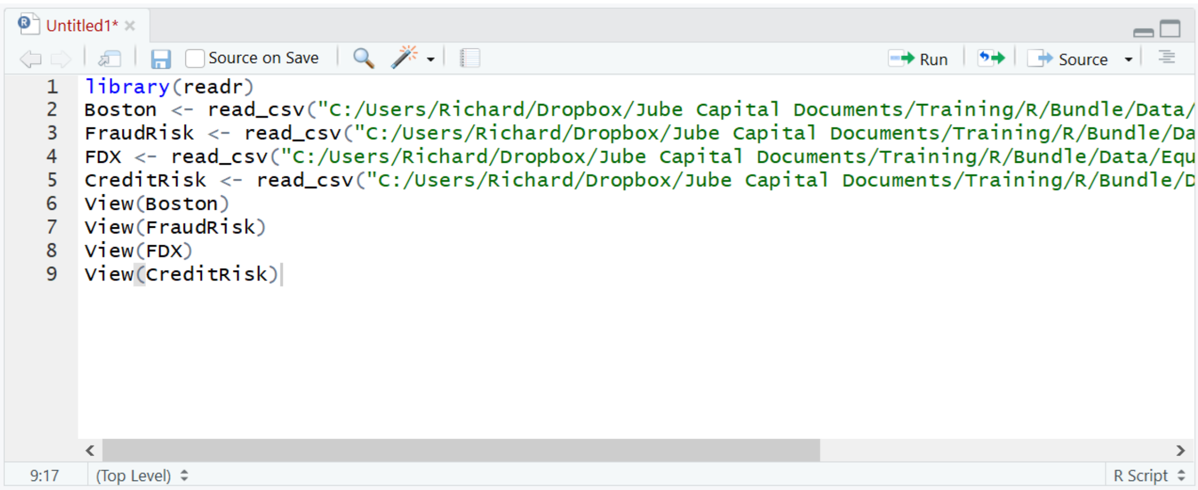
It should be noted that R, in our orbit, is predominately used for the rapid exploration of data and the creation of models only and these procedures focus only on what is adequate to achieve that aim.

The following datasets are going to be used in this module:

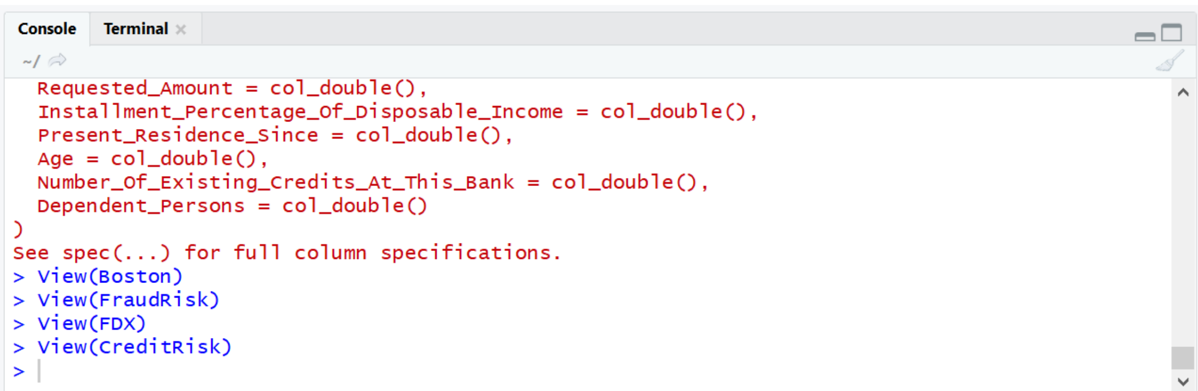
- BostonHousing.csv.
- CreditRisk.csv.
- FDX.csv (FexEx Stock Prices)
- FraudRisk.csv

Before beginning these procedures, ensure that each of the files has been loaded with the following script:

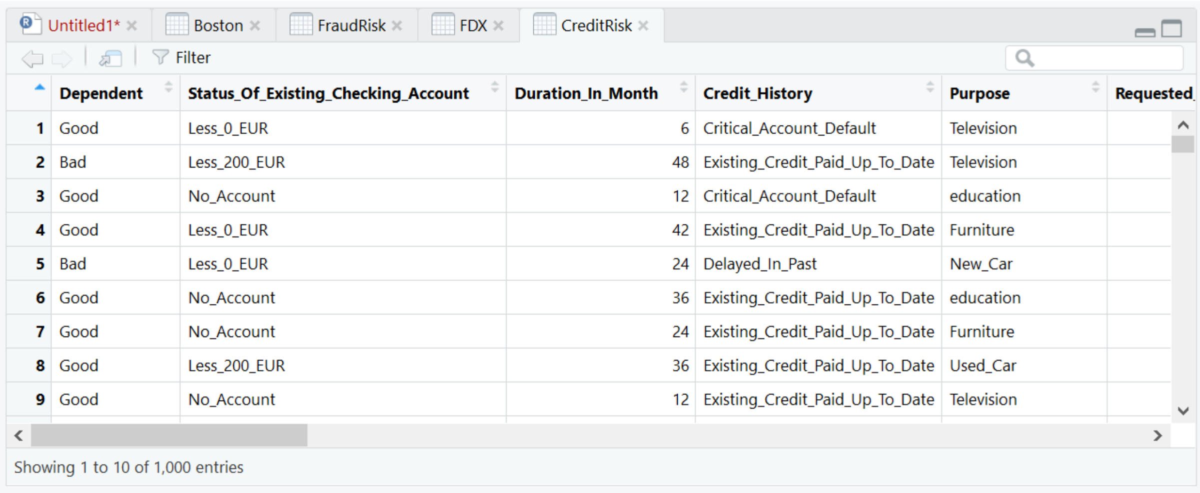
```
library(readr)
Boston <- read_csv("C:/Users/Richard/Dropbox/Jube Capital
Documents/Training/R/Bundle/Data/Boston/Boston.csv")
FraudRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital
Documents/Training/R/Bundle/Data/FraudRisk/FraudRisk.csv")
FDX <- read_csv("C:/Users/Richard/Dropbox/Jube Capital
Documents/Training/R/Bundle/Data/Equity/Equity/FDX.csv")
CreditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital
Documents/Training/R/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)View(Boston)
```



Run the block of script to console:

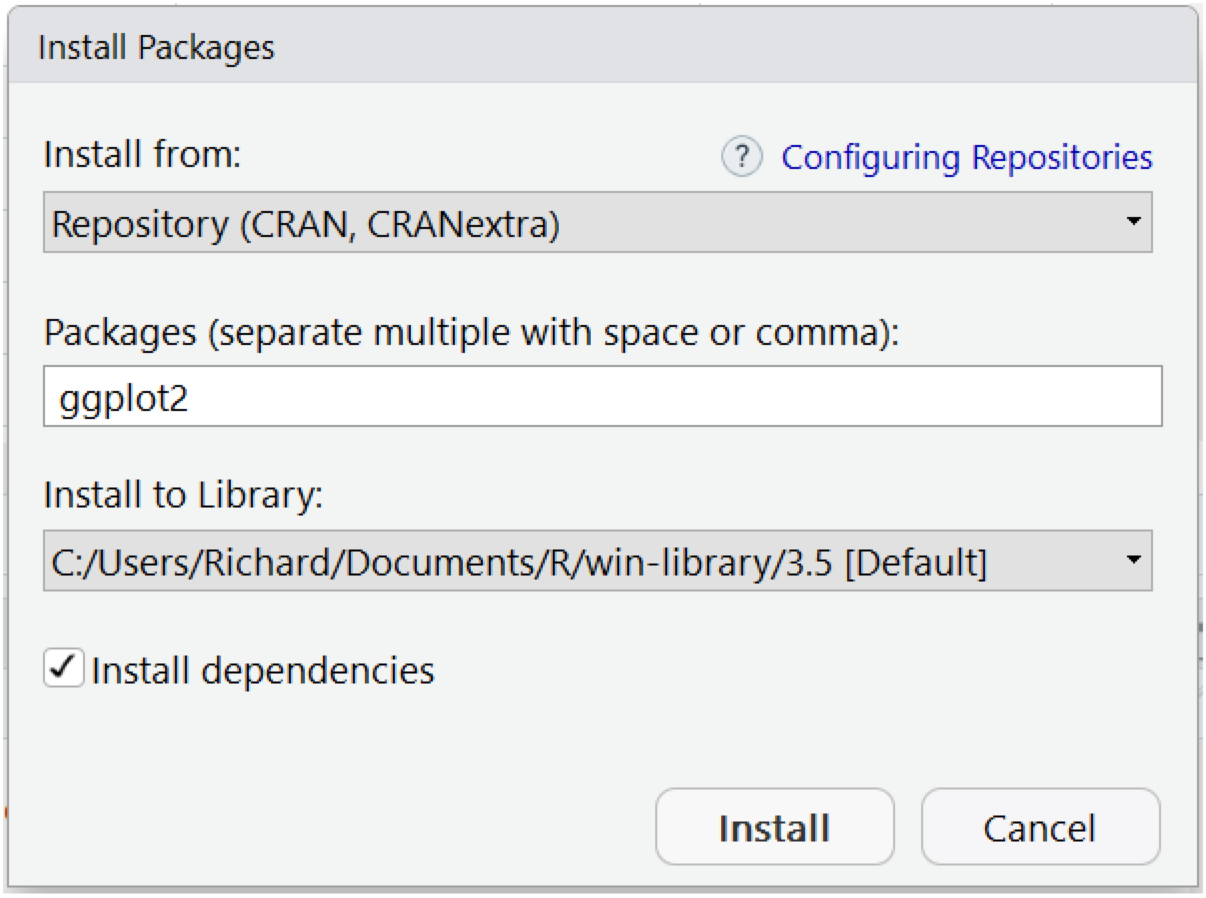


The required datasets will be loaded into the R session and displayed in tabs as a consequence of the View() function being recalled on each data frame:

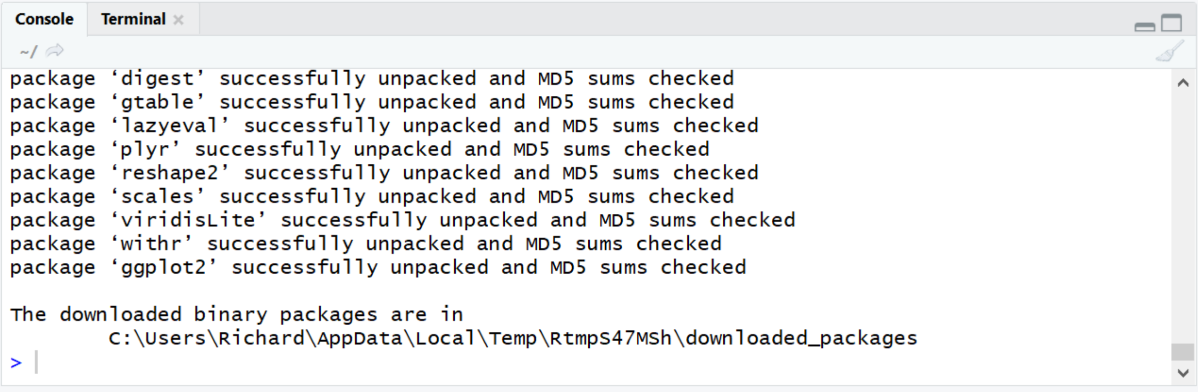


	Dependent	Status_Of_Existing_Checking_Account	Duration_In_Month	Credit_History	Purpose	Requested
1	Good	Less_0_EUR	6	Critical_Account_Default	Television	
2	Bad	Less_200_EUR	48	Existing_Credit_Paid_Up_To_Date	Television	
3	Good	No_Account	12	Critical_Account_Default	education	
4	Good	Less_0_EUR	42	Existing_Credit_Paid_Up_To_Date	Furniture	
5	Bad	Less_0_EUR	24	Delayed_In_Past	New_Car	
6	Good	No_Account	36	Existing_Credit_Paid_Up_To_Date	education	
7	Good	No_Account	24	Existing_Credit_Paid_Up_To_Date	Furniture	
8	Good	Less_200_EUR	36	Existing_Credit_Paid_Up_To_Date	Used_Car	
9	Good	No_Account	12	Existing_Credit_Paid_Up_To_Date	Television	

All of the procedures as follows make use of the ggplot2 package, hence it is necessary to install the ggplot2 package using RStudio:

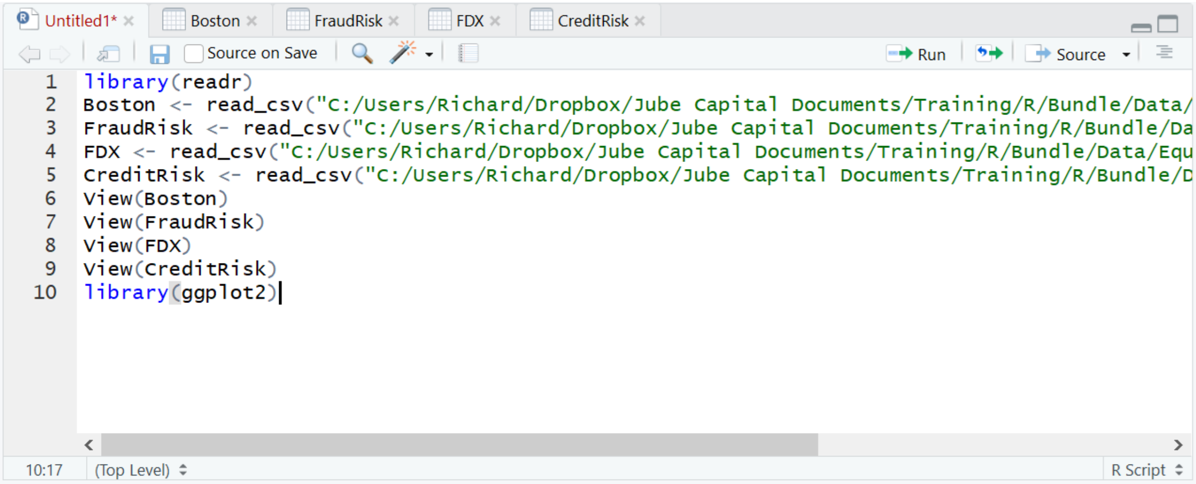


Clicking install will download and install the package:

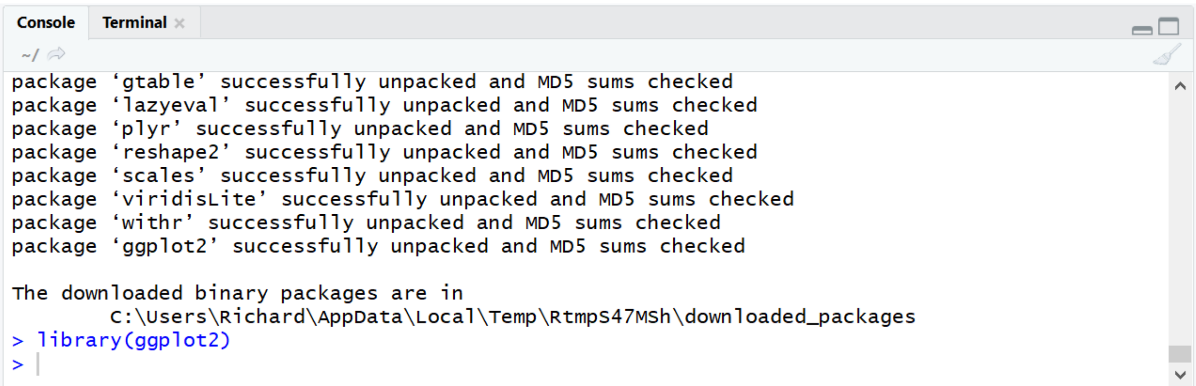


To reference the library:

library(ggplot2)



Run the line of script to console:

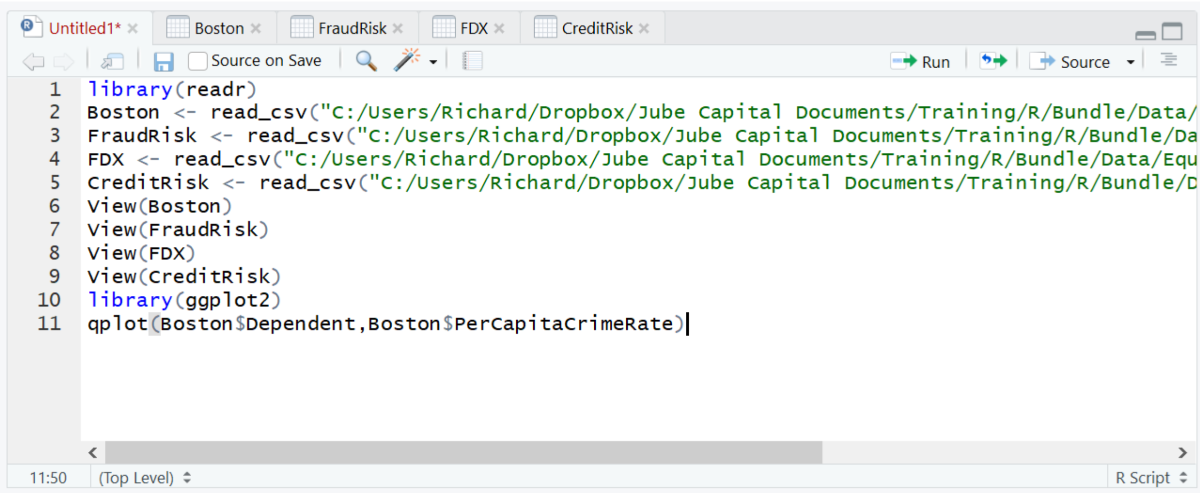


The RStudio environment is now configured to use the qplot() function and other ggplot2 package functions.

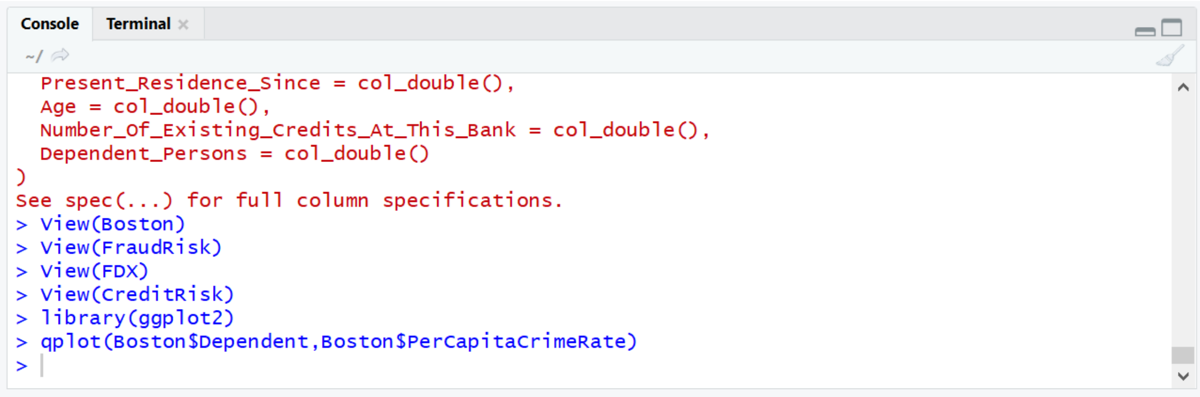
Procedure 1: Quickly Creating a Scatter Plot with qplot()

The following procedure will show how the QPlot() functon can be used in a similar manner as the plot() function to create a scatter chart. To create a scatter plot that compares the PerCapitaCrimeRate to the House prices in the Boston Housing dataset:

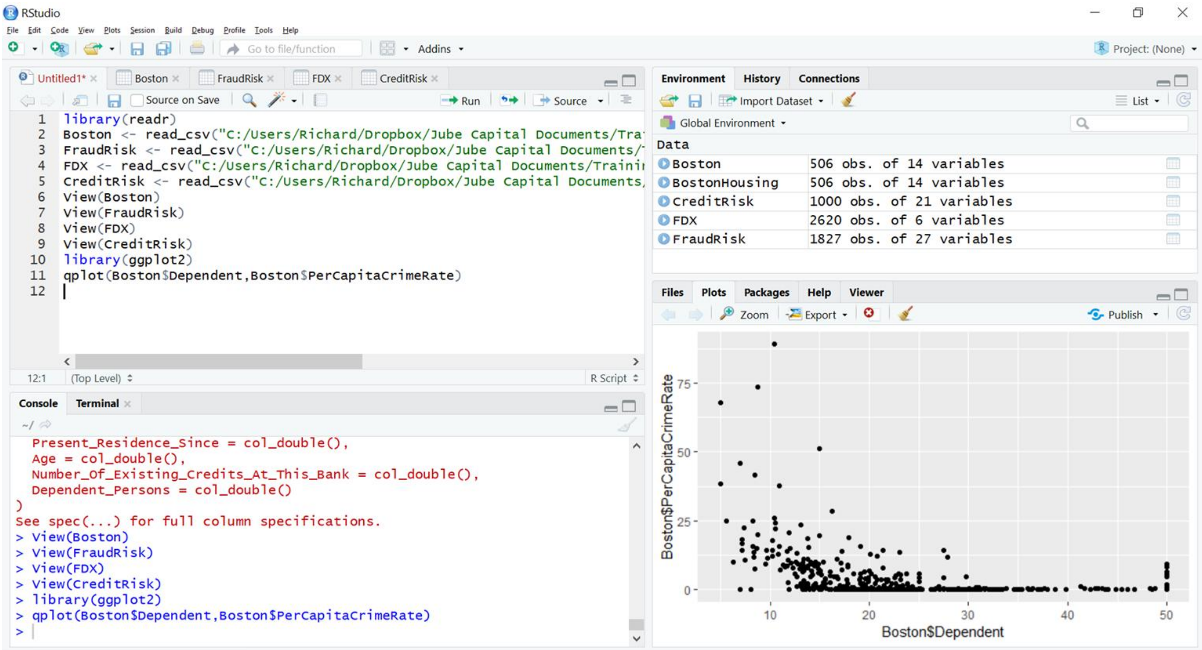
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)



Run the line of script to console:



It can be seen that the plot is available in RStudio:

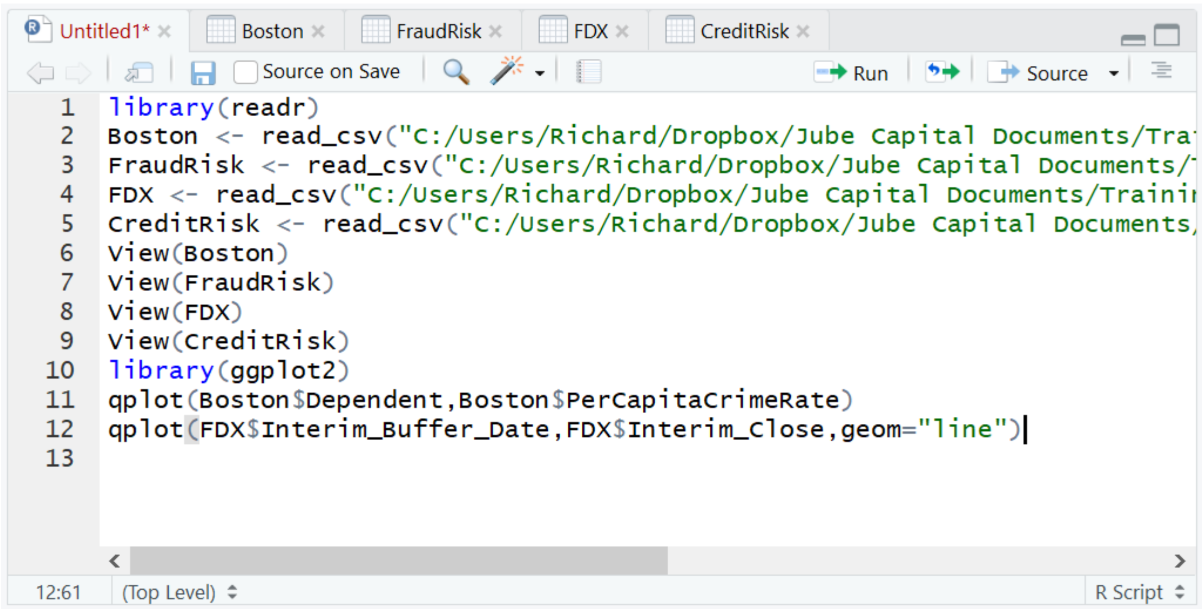


The plot bears stark resemblance to the product of the base R graphics plot() function, except the rendering quality is of better quality. This is common for all of the charts explained as follows.

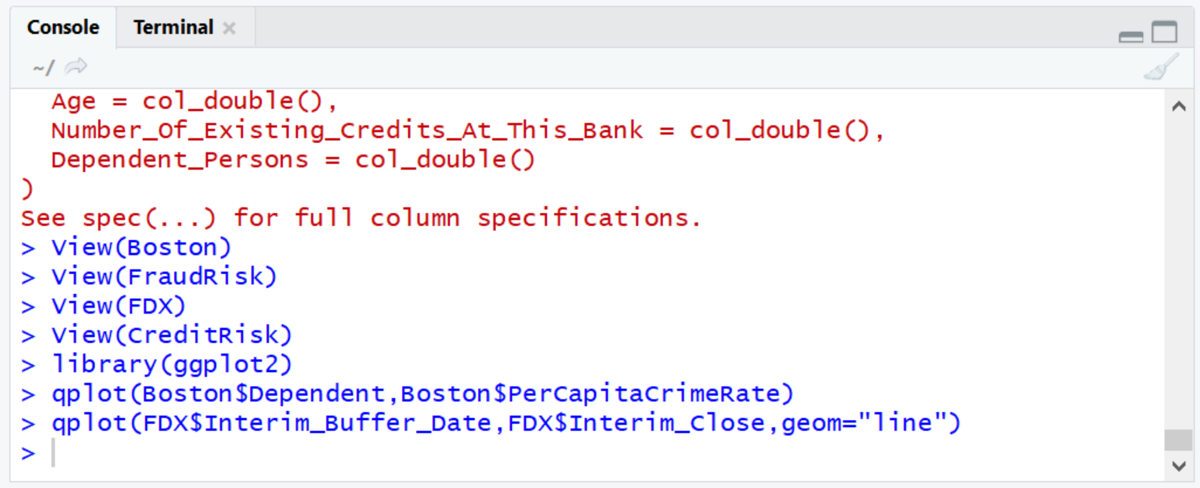
Procedure 2: Quickly Creating a Line Chart with qplot()

Displaying a stock price over time is a convenient example to show the functionality of qplot when used to make a line chart. In this example, the FedEx stock price is going to be plotted over time. As with the scatter plot example, the qplot function takes two vectors, however, the geom parameter will be used to specify the type of chart, in this case "line". To create a time series line chart, pass a date (Interim_Date) and the price (Interim_Close):

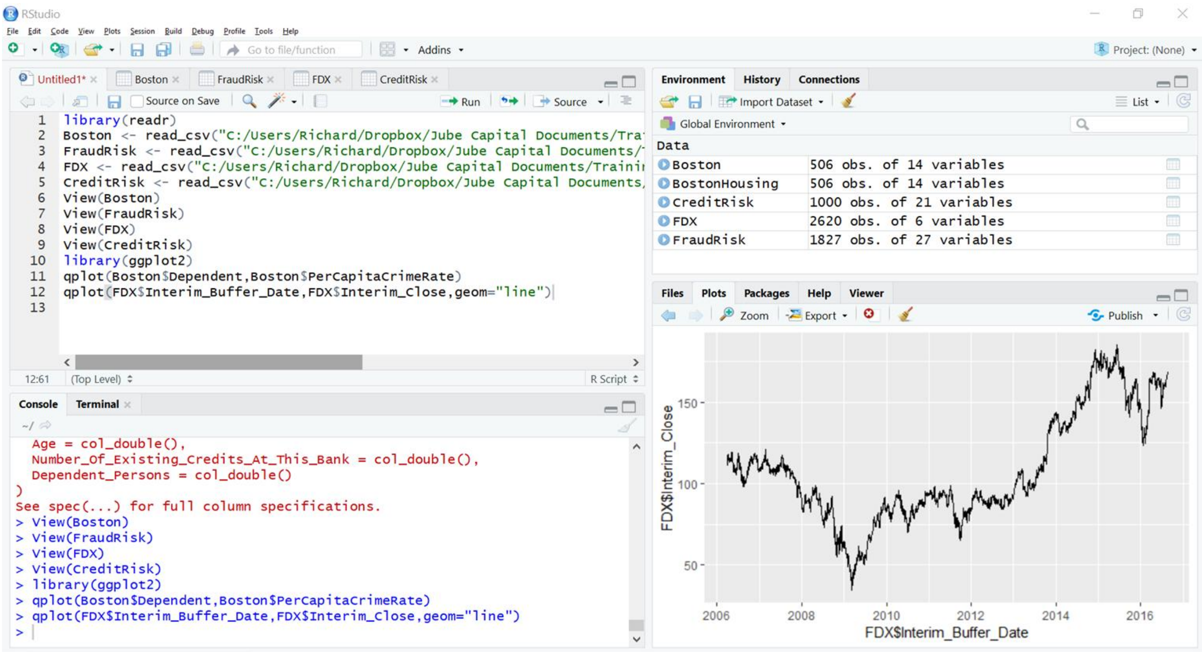
```
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
```



Run the line of script to console:



It can be observed that the chart has been rendered to the plots section of RStudio:



This procedure should exhibit that qplot has default of a scatter chart, however it can be easily changed to other types by varying the geom parameter.

Procedure 3: Quickly Creating a Bar Chart with qplot()

To create a bar chart in Base R it is necessary to perform some preaggregation of values. A useful function, used extensively in subsequent procedures, is the table() function. The table() function will scan a vector and allocate counts for the distinct values available in that vector.

In this example the CreditRisk dataset is going to be used to present a bar chart of the frequency of each loan purpose. Firstly, create a table of the loan purpose to show the original method of bar chart creation and the functionality of the table() function:

Purpose <- table(CreditRisk$Purpose)

```r
library(readr)
Boston <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Tra
FraudRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/
FDX <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Trainin
CreditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents,
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)
library(ggplot2)
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
table(CreditRisk$Purpose)
```

Run the line of script to console:

```r
library(readr)
Boston <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Tra
FraudRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/
FDX <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Trainin
CreditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents,
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)
library(ggplot2)
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
Purpose <- table(CreditRisk$Purpose)
```

Write the table to console:

Purpose

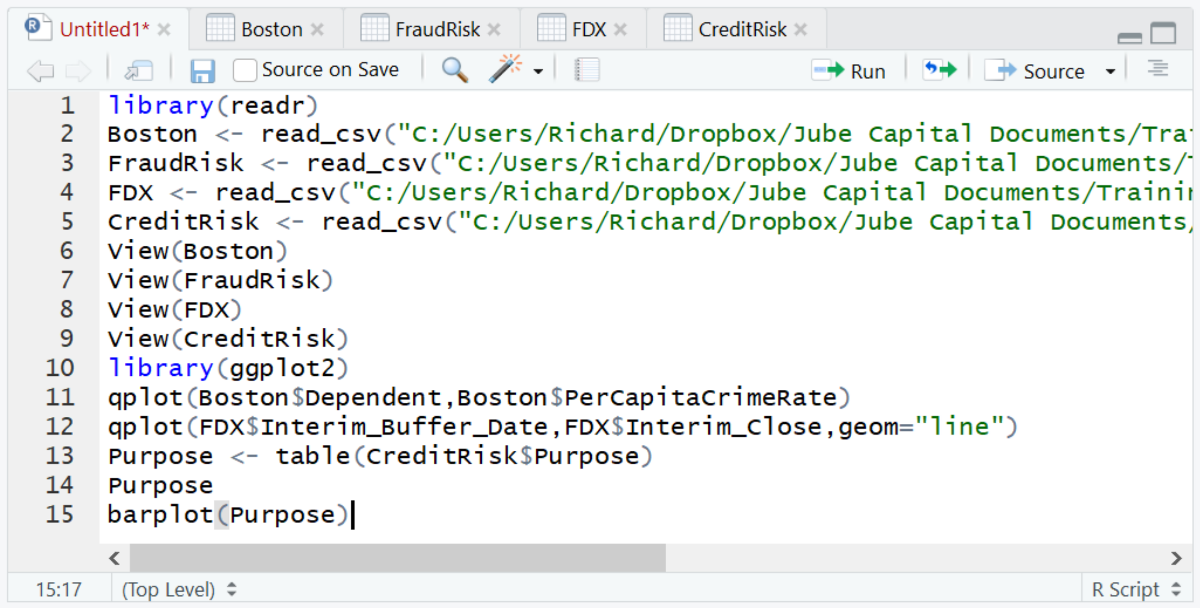
```
library(readr)
Boston <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Tra
FraudRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/
FDX <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Traini
CreditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents,
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)
library(ggplot2)
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
Purpose <- table(CreditRisk$Purpose)
Purpose
```

Run the line of script to console:

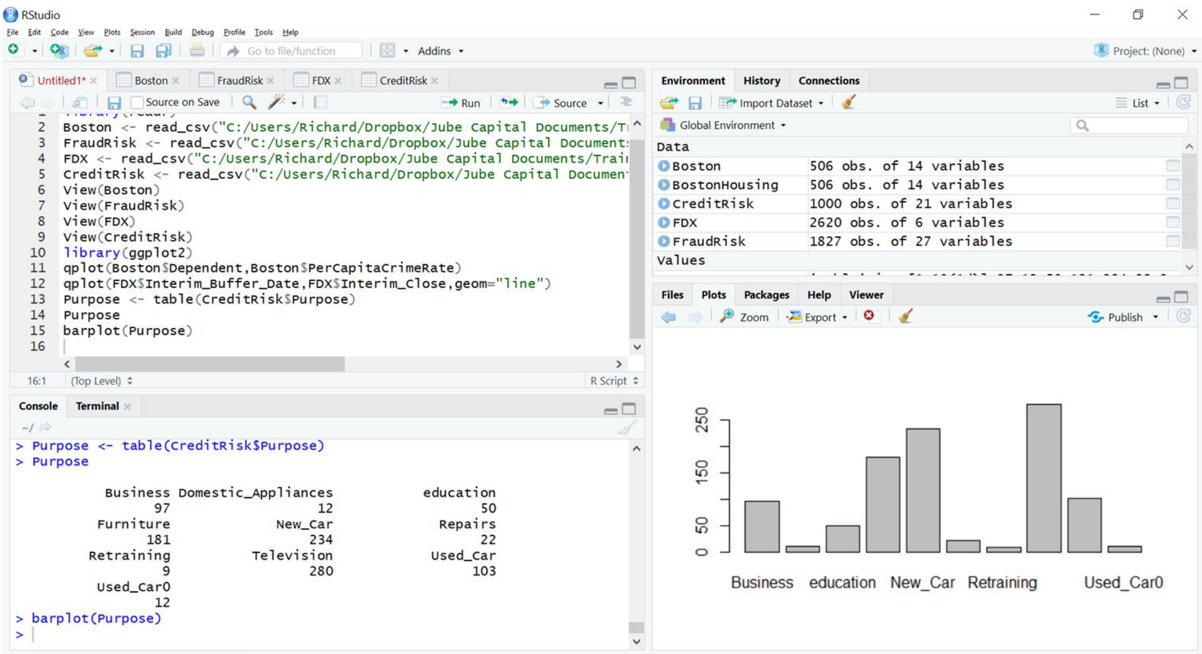
```
> qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
> Purpose <- table(CreditRisk$Purpose)
> Purpose

          Business Domestic_Appliances           education 
                97                  12                  50 
         Furniture             New_Car             Repairs 
               181                 234                  22 
        Retraining          Television            Used_Car 
                 9                 280                 103 
          Used_Car0 
                12 
> 
```

It can be observed that the frequencies have been apportioned next to the loan purpose vector. This table can then be passed to the base R function barplot():

barplot(Purpose)



Run the line of script to console:



It can be seen that the bar chart has been written out to the plots area of RStudio. Using qplot() it is however possible for the aggregation to take place by simply passing two vectors in the same manner as a scatter plot, simply specifying the geom parameter to be "bar":

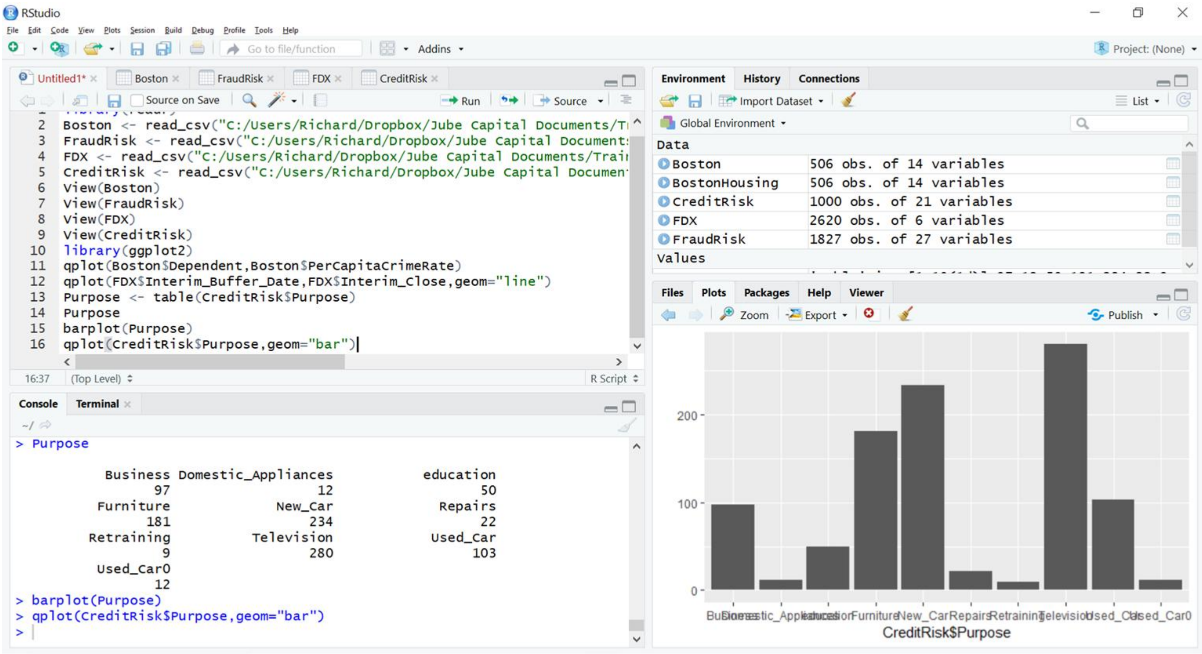
qplot(CreditRisk$Purpose,geom="bar")

```
FraudRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Document
FDX <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documents/Trai
CreditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documen
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)
library(ggplot2)
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
Purpose <- table(CreditRisk$Purpose)
Purpose
barplot(Purpose)
qplot(CreditRisk$Purpose,geom="bar")
```

Run the line of script to console:

```
> Purpose

         Business Domestic_Appliances           education
               97                  12                  50
        Furniture             New_Car             Repairs
              181                 234                  22
       Retraining          Television            Used_Car
                9                 280                 103
        Used_Car0
               12
> barplot(Purpose)
> qplot(CreditRisk$Purpose,geom="bar")
>
```

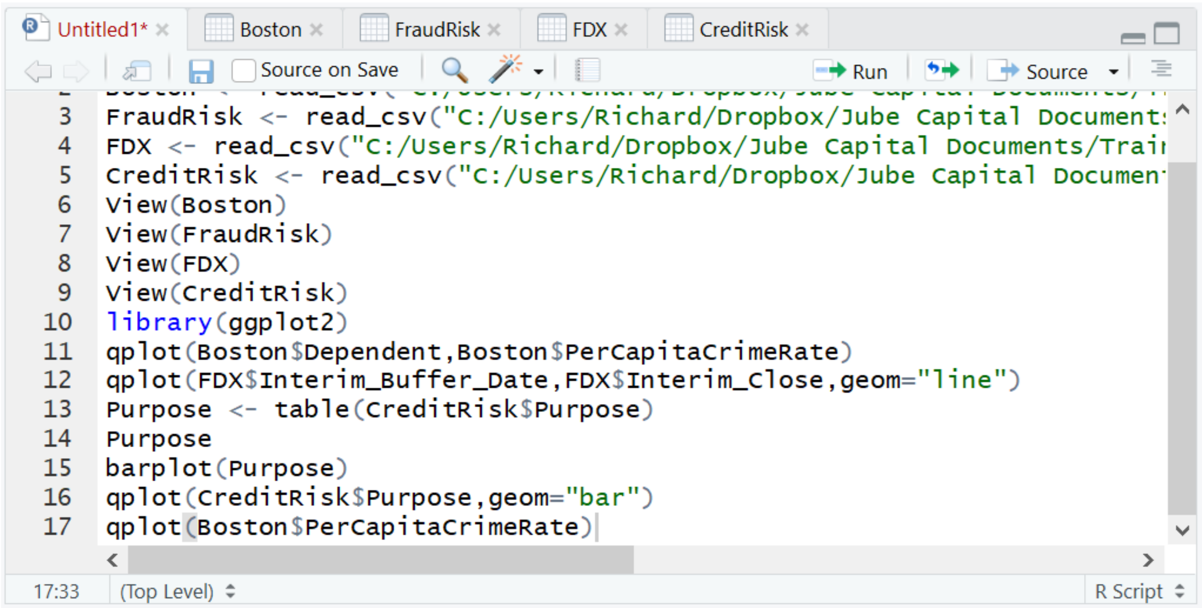
In the RStudio plots pane it can be seen that a bar chart has been created without the need to aggregate using a table:



Procedure 4: Quickly Creating a Histogram with qplot()

The qplot() histogram bears resemblance to the hist() function, being called almost identically:

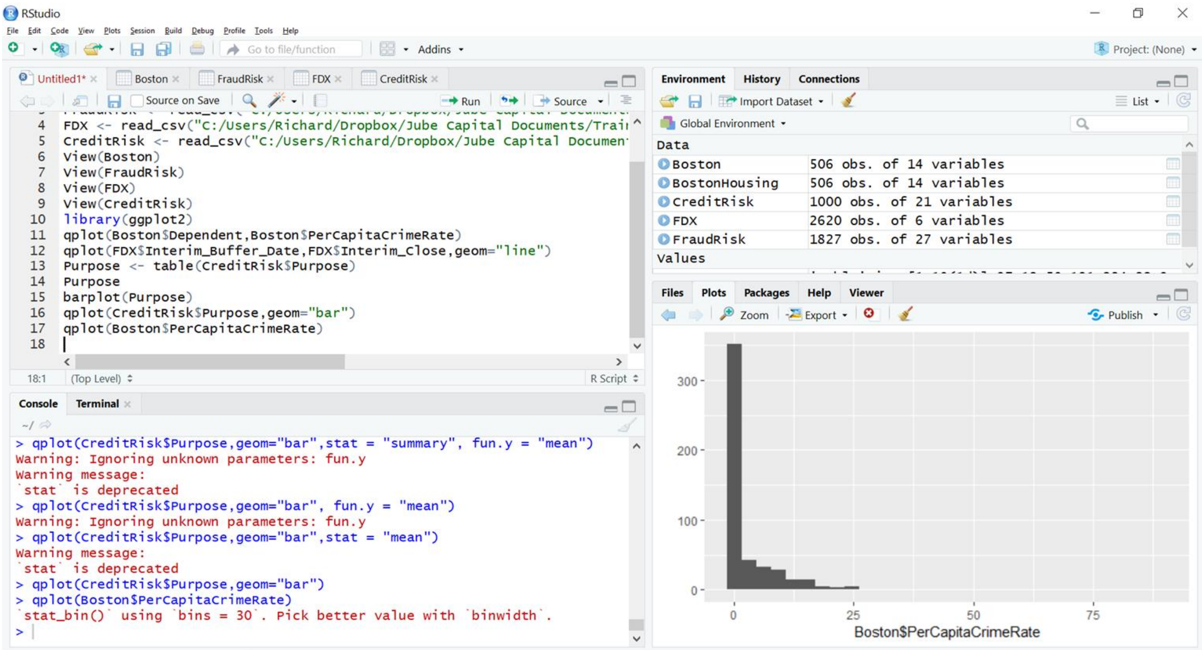
qplot(Boston$PerCapitaCrimeRate)



Run the line of script to console:

```
> qplot(CreditRisk$Purpose,geom="bar",stat = "summary", fun.y = "mean")
Warning: Ignoring unknown parameters: fun.y
Warning message:
`stat` is deprecated
> qplot(CreditRisk$Purpose,geom="bar", fun.y = "mean")
Warning: Ignoring unknown parameters: fun.y
> qplot(CreditRisk$Purpose,geom="bar",stat = "mean")
Warning message:
`stat` is deprecated
> qplot(CreditRisk$Purpose,geom="bar")
> qplot(Boston$PerCapitaCrimeRate)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> 
```

It can be seen that an error message has been created suggesting that the bin width is too wide, which is clearly the case in the plot being written out with a very wide scale:



Specifying the binwidth parameter of the qplot function solves the issue of their being too many bins by widening the size of the bins:

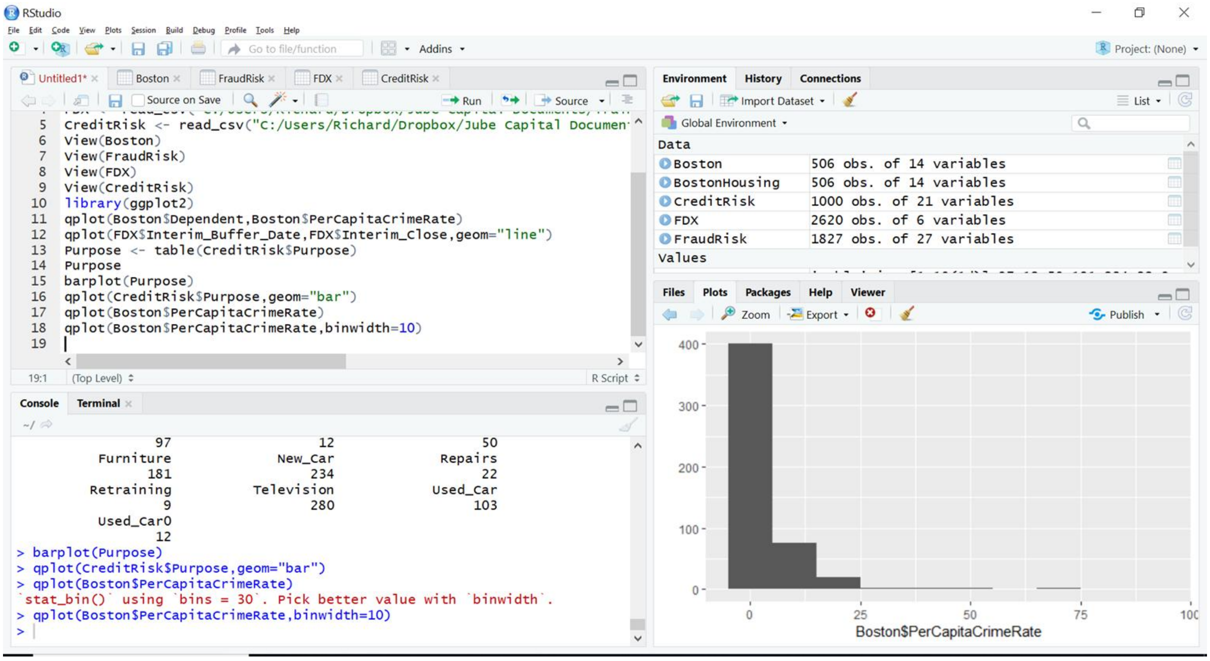
qplot(Boston$PerCapitaCrimeRate,binwidth=10)

```
creditRisk <- read_csv("C:/Users/Richard/Dropbox/Jube Capital Documen
View(Boston)
View(FraudRisk)
View(FDX)
View(CreditRisk)
library(ggplot2)
qplot(Boston$Dependent,Boston$PerCapitaCrimeRate)
qplot(FDX$Interim_Buffer_Date,FDX$Interim_Close,geom="line")
Purpose <- table(CreditRisk$Purpose)
Purpose
barplot(Purpose)
qplot(CreditRisk$Purpose,geom="bar")
qplot(Boston$PerCapitaCrimeRate)
qplot(Boston$PerCapitaCrimeRate,binwidth=10)
```

Run the line of script to console:

```
             97                12                50
      Furniture           New_Car           Repairs
            181               234                22
     Retraining        Television          Used_Car
              9               280               103
      Used_Car0
             12
> barplot(Purpose)
> qplot(CreditRisk$Purpose,geom="bar")
> qplot(Boston$PerCapitaCrimeRate)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> qplot(Boston$PerCapitaCrimeRate,binwidth=10)
>
```

It can be seen that a histogram has been plotted in RStudio, with fewer bars owing to the distances for the bars being wider:

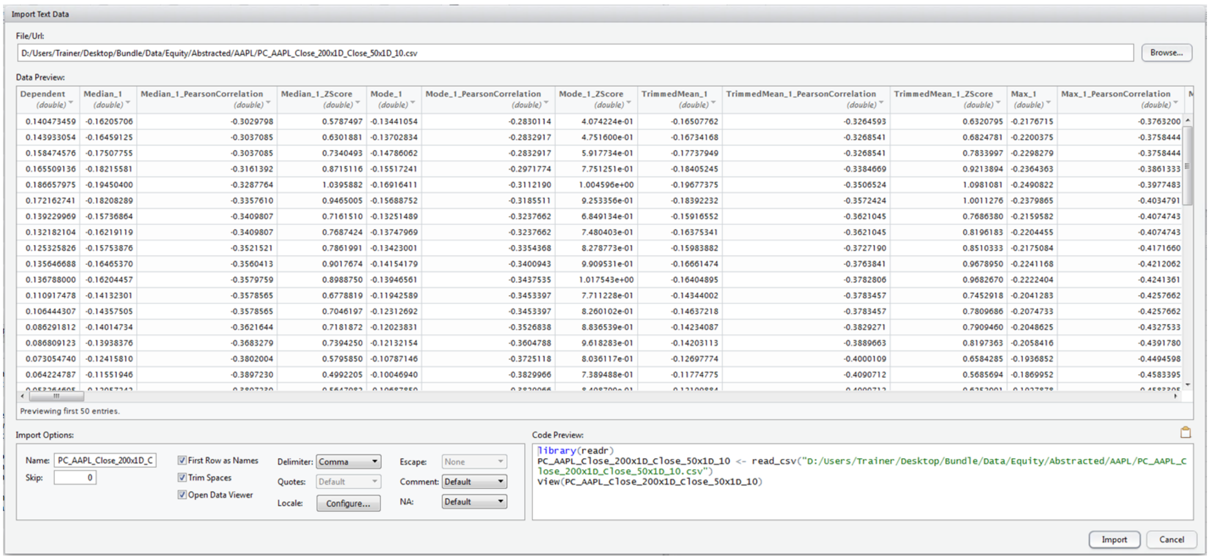


Module 8: Linear Regression.

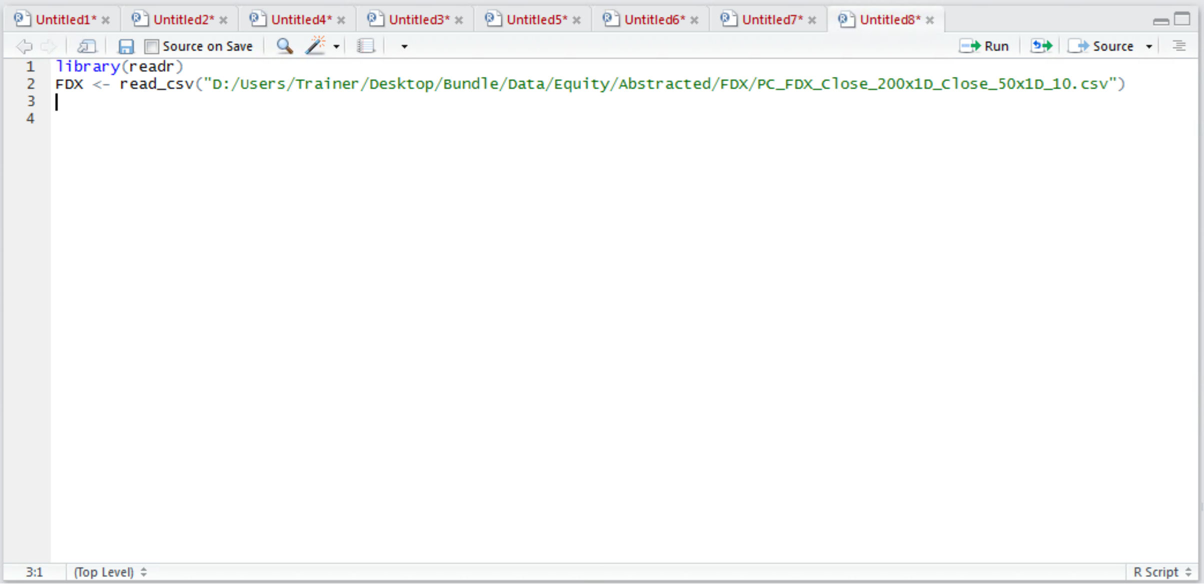
Linear Regression is a modelling technique that can be used for numeric prediction where the values are fairly normal in distribution.

The dataset that is used in this module is available under Bundle\Data\Equity\Abstracted\FDX\PC_FDX_Close_200x1D_Close_50x1D_10.csv which contains data that has already been abstracted for the FedEx stock on the NYSE.

To proceed with the subsequent procedures, it is necessary to import the file PC_FDX_Close_200x1D_Close_50x1D_10.csv into R as per procedure 19:

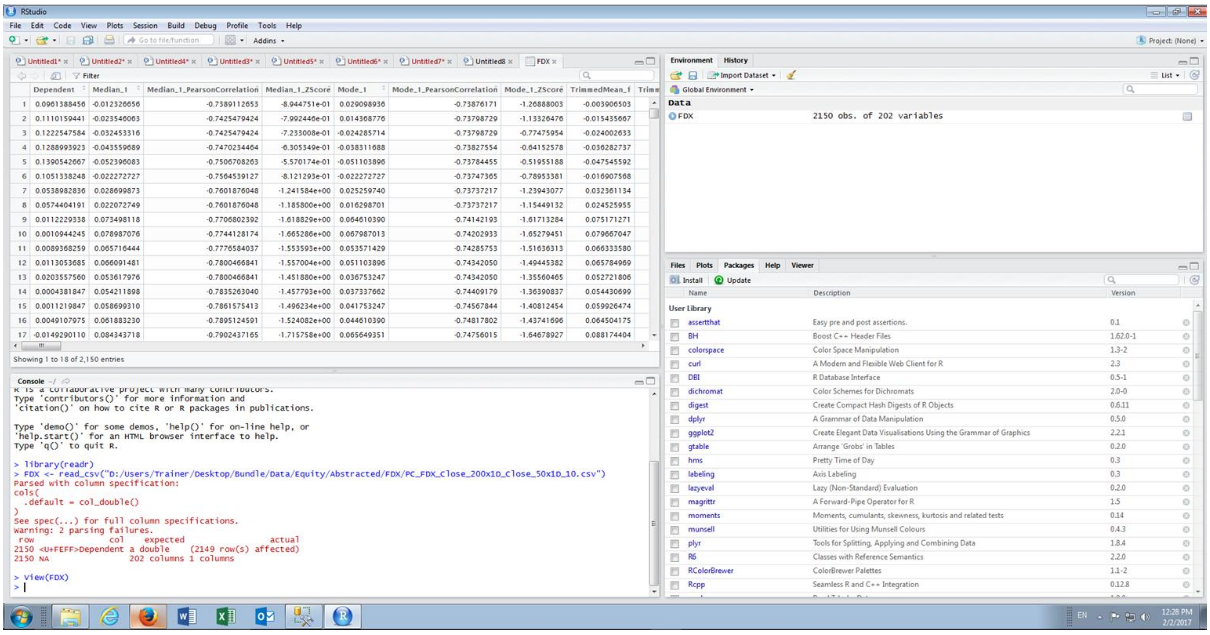


For completeness the library(readr) and Load_CSV() function text will be copied to the curret script to ensure that the script remains portable:



For ease and simplicity the name of the data set has been changed to FDX from the default of PC_FDX_Close_200x1D_Close_50x1D_10.csv:

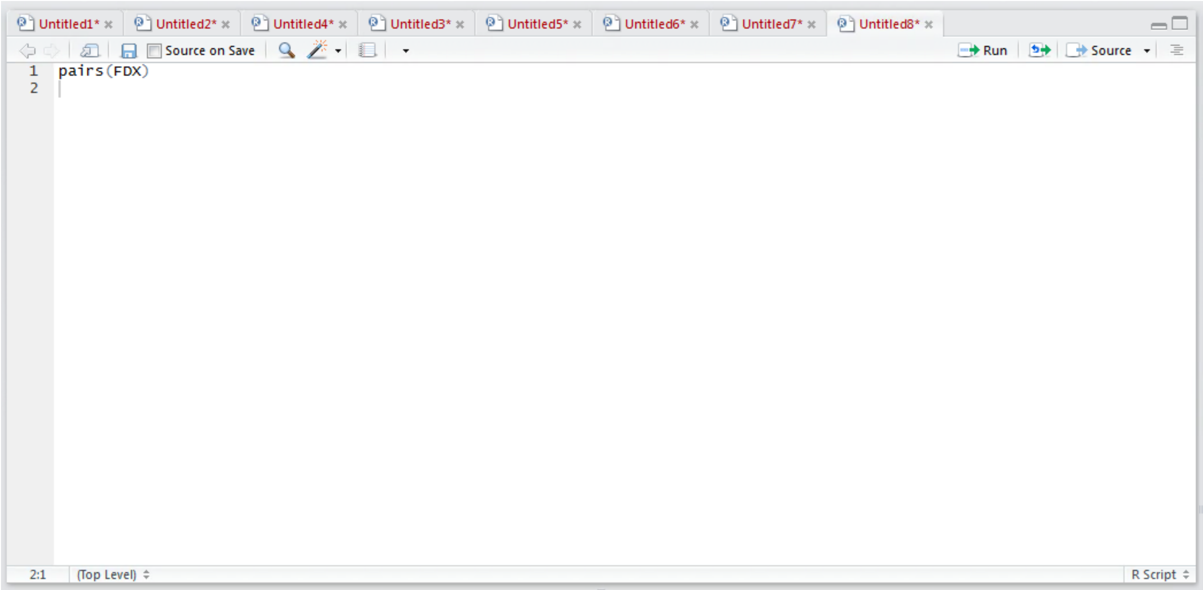
Executing the load, the contents of the csv file will automatically be exposed on invoking the view() function in the console:



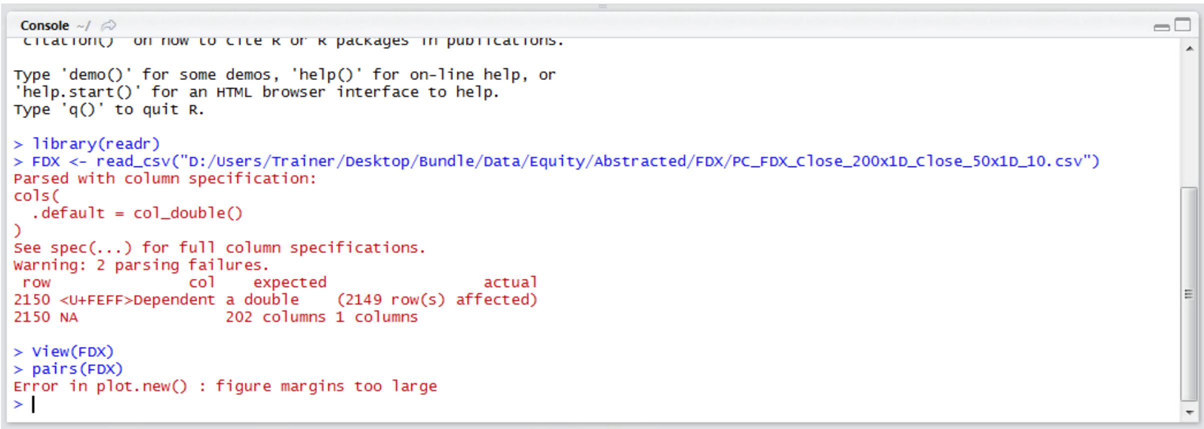
Procedure 1: Scanning Scatter Plots for Relationships.

R has a function called pairs() which is incredibly useful for visualizing the relationships existing between variables inside a data frame on a fairly exhaustive basis. It is possible to simply pass the data frame as an argument to the pairs function for an exhaustive visualization to be produced:

pairs(FDX)

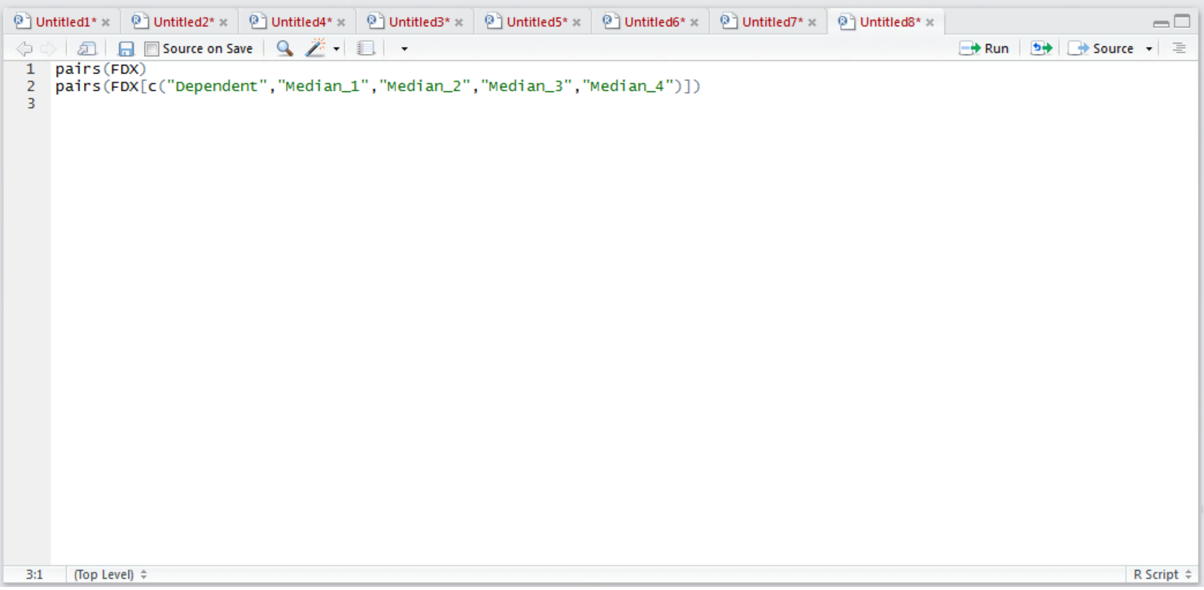


Run the line of script to console:

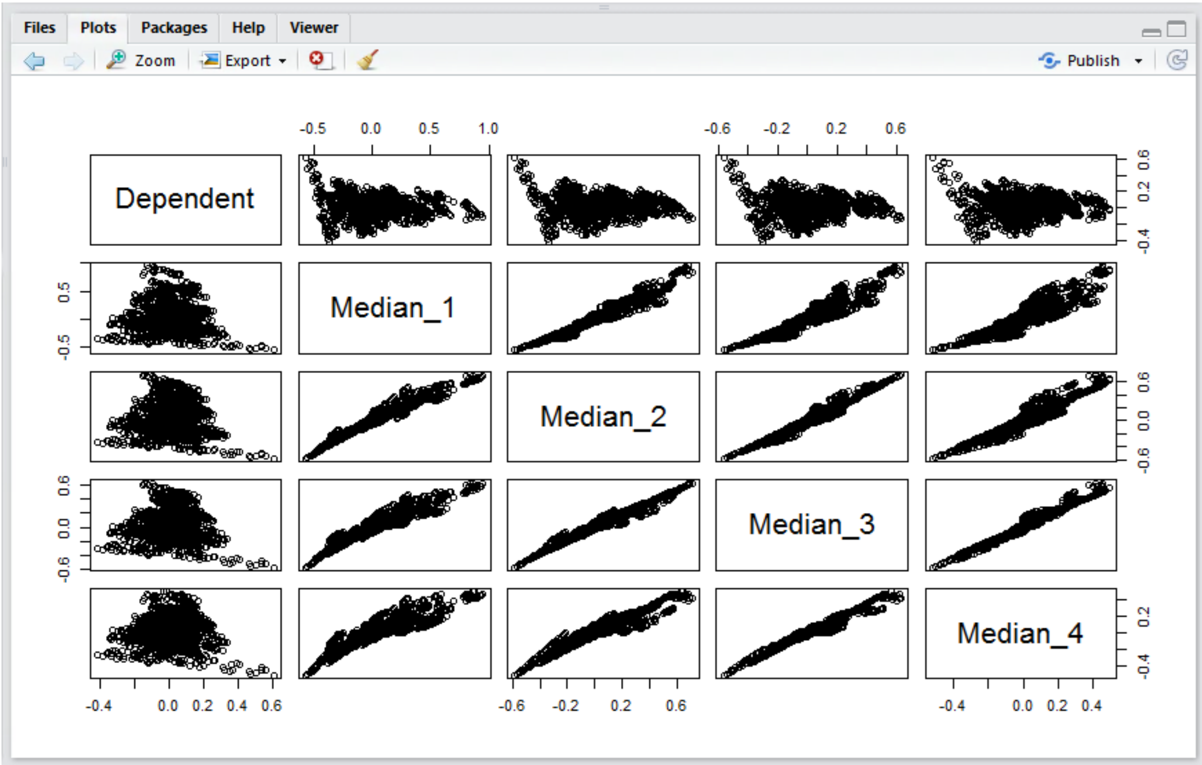


In this example, the data frame is far too large, having hundreds of columns, which would create a visualization that is many times larger than the RStudio plots pane. It follows that more selectivity in the vectors to be used in the visualization need be mustered, a simple matter of subscripting the data frame using square brackets as an argument to the Pairs function:

pairs[c("Dependent"," Median_1"," Median_1_PearsonCorrelation"," Median_1_ZScore "," Mode_1"," Mode_1_PearsonCorrelation","Mode_1_ZScore")]



Run the line of script to console to produce a matrix of scatter plots:



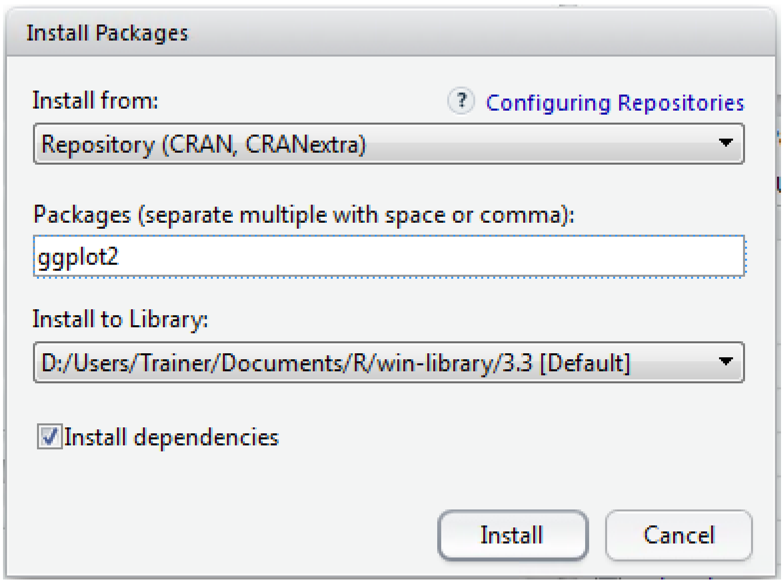
In this example, the relationship between the dependent variable and the independent variables is most interesting, at a moment's glance it can be seen that several extreme relationships exist.

This process would be repeated, including the dependent variable, for several other groups of independent variables until such time as a familiarity of relationships has been amassed and a good feel for how independent variables relate to the dependent variable has been obtained. This process can help identify independent variables that correlate well with the dependent variable, carrying these variables forward for the purposes of modeling.

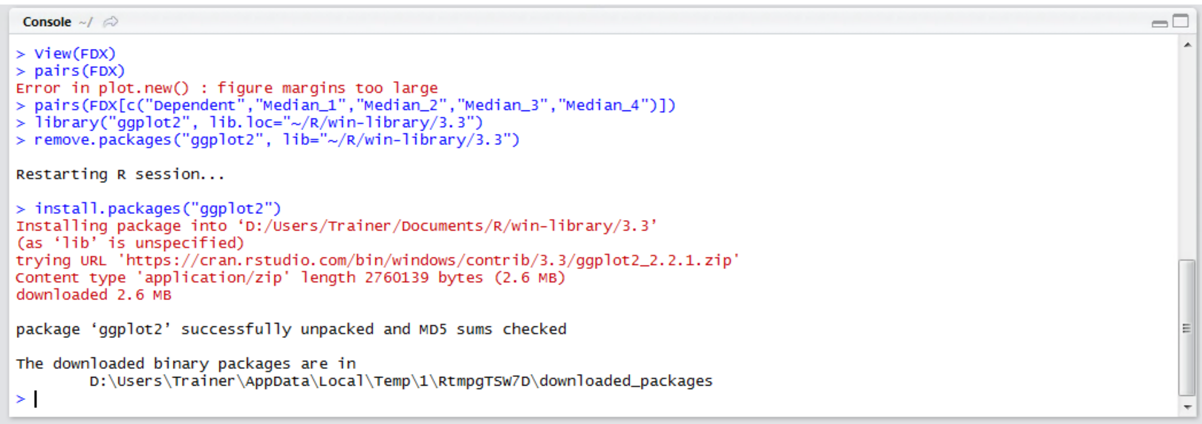
Procedure 2: Creating a Scatter Plot for Closer Inspection with ggplot2.

The scatter plot matrix created in procedure is an extremely useful and informative tool, if lacking beauty. A package that cannot escape mention for the creation of graphics in R is ggplot2, which is a powerful and flexible graphics package for creating charts and visualisations every bit as beautiful as that which could be found in Excel.

Start by installing the ggplot package using RStudio and as described in procedure 9:

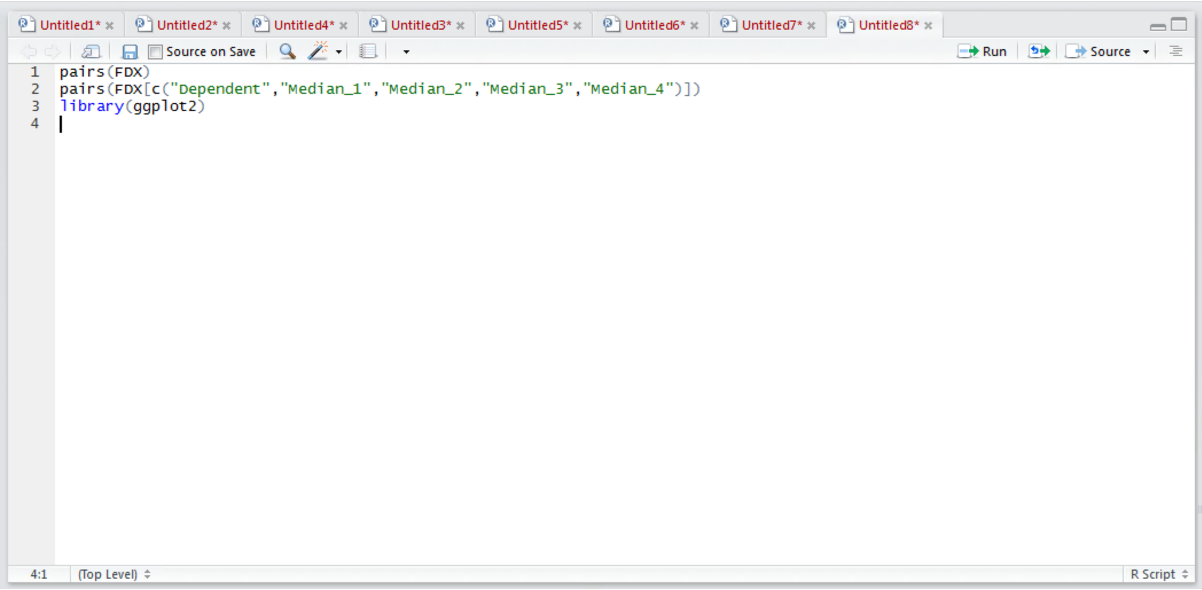


Clicking install to download and install the package:

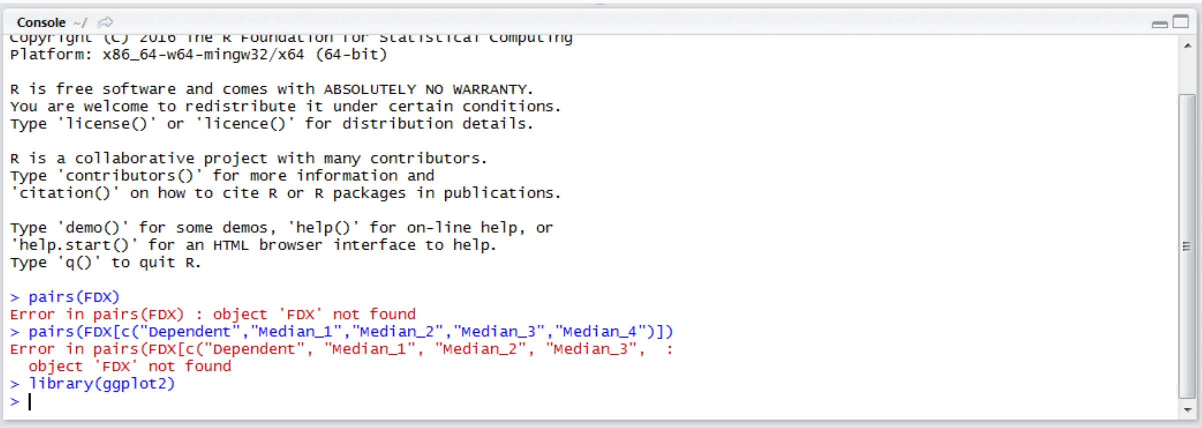


Once the packages has been downloaded and installed, reference the package using the library() function and its name ggplot2:

library(ggplot2)

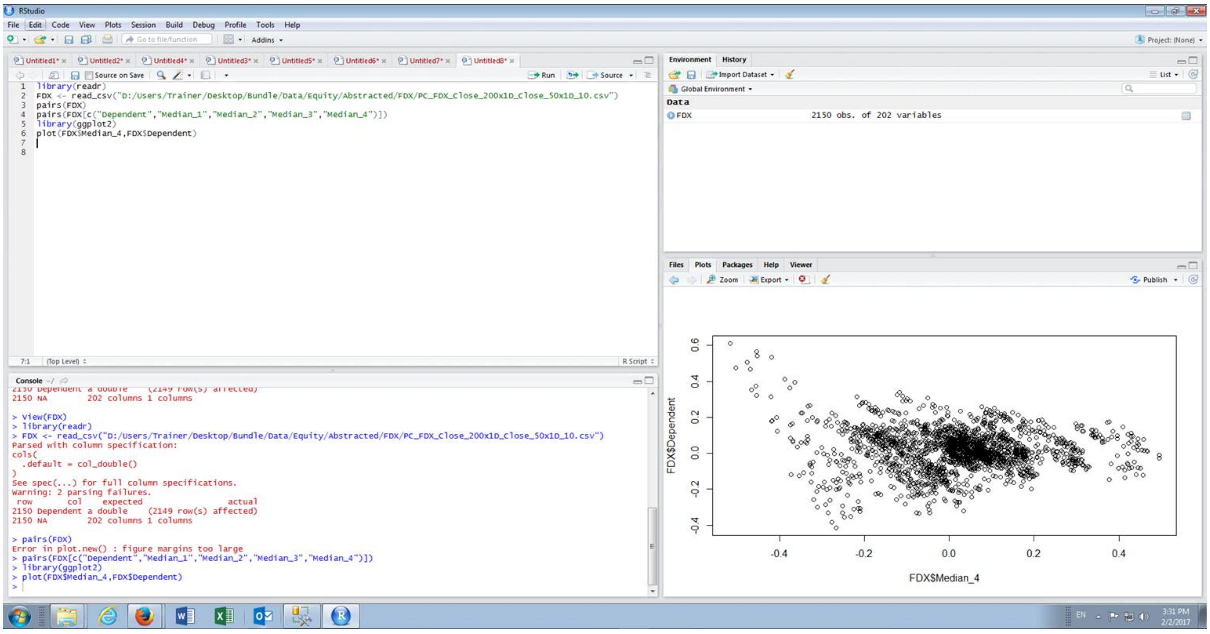


Run the line of script to console:



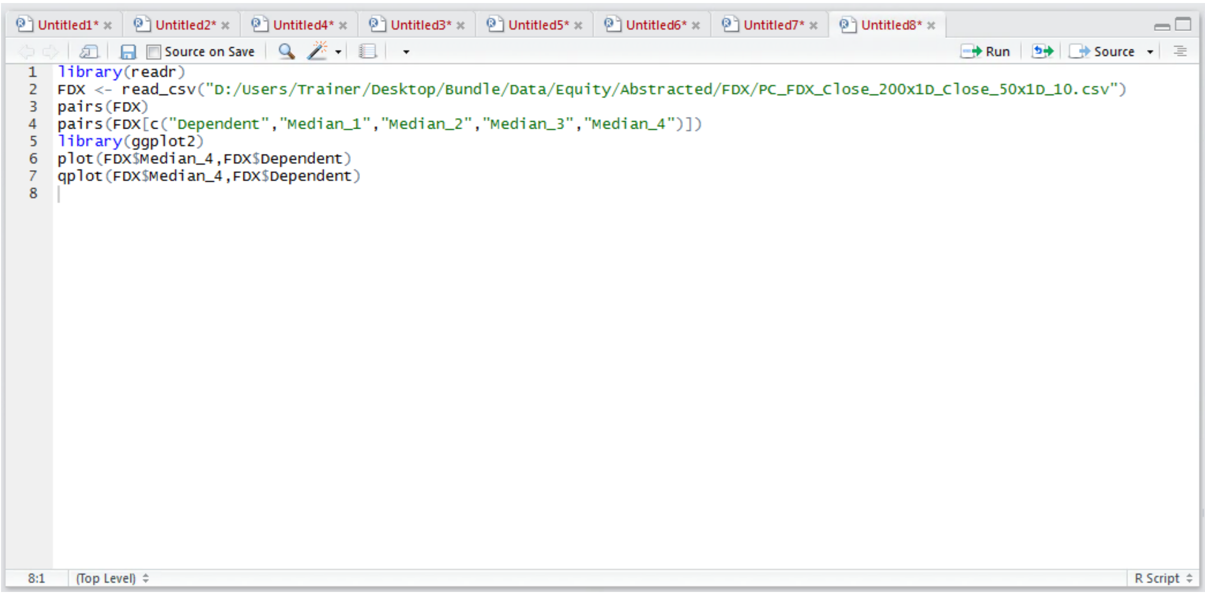
In this example a scatter plot will be created with the Dependent Vector on the y Axis and the Median_4 on the x axis, and initially using just the built in function plot():

plot(FDX$Median_4,FDX$Dependent)

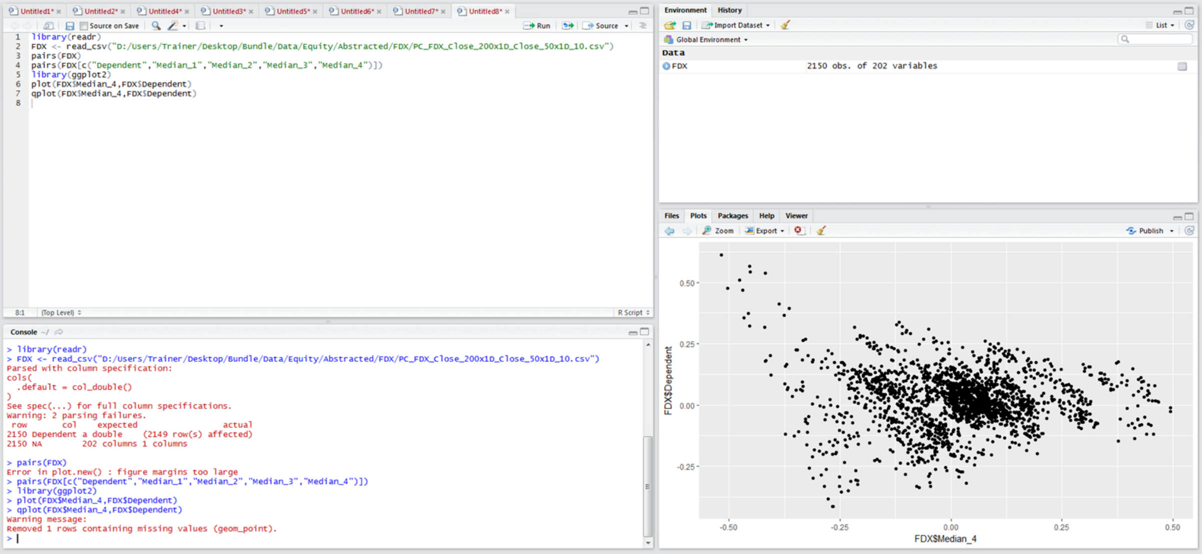


The signature of the plot() function is effortless and it is a fantastic extensions to perform quick and exploratory data analysis, although it may not be visually impressive enough for the purposes of presentations. qplot() is a function in the ggplot2 package and achieves much the same, just visually more striking:

qplot(FDX$Median_4,FDX$Dependent)



Run the line of script to console:



The package ggplot2 provides a plethora of functions that will create rich and visually impressive graphics, from the being able to manipulate colours to correctly titling a plot with the intention of creating graphics fit for publishing.

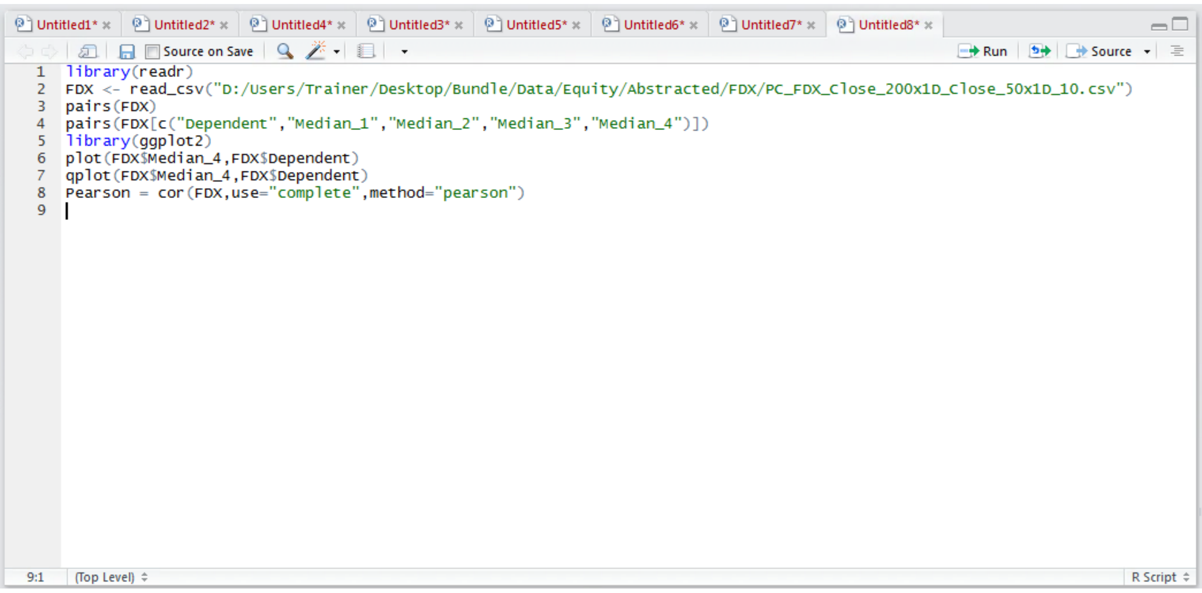
The ggplot functionality will be steadily introduced in subsequent procedures although creating visually striking charts for publication is outside the scope of this course.

Procedure 3: Create a Correlation Matrix using Spearman and Pearson.

Correlation is a measure of relationship and direction of that relationship. It is a single value that ranges from -1 to +1, which would signal the direction and strength of a relationship. Both -1 and +1 are, in their extremes, equally interesting. A correlation matrix takes all the variables together and produces the correlation value, the strength of their relationship in one director of another, between each variable.

The matrix will be the foundation for many of the techniques used in the following procedures. In R the cor() function is used to produce correlation matrices upon data frames. To create a Pearson correlation matrix:

Pearson = cor(FDX,use="complete",method="pearson")

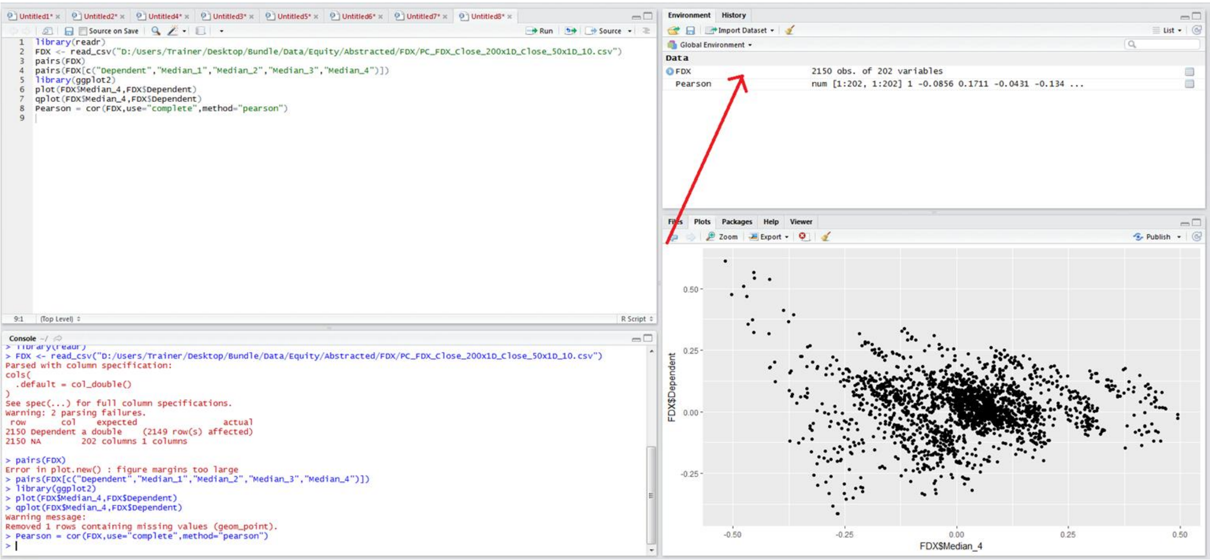


It can be seen that the cor() function takes the FDX data frame as its source. The method argument specifies which type of correlation calculation to perform, an alternative would be "spearman".

Lastly "use" argument tells the cor() function how to deal with missing or bad data, whereby the default is to throw an error, hence it is a good idea to specify "complete" when working with very large datasets else it is likely the entire matrix would be returned as "NA".

Run the line of script to console:

It can be seen that a matrix by the name of Pearson has been created and is available in the environment pane:



Clicking on the entry in the environment pane would expand a view panel and display a more visually satisfying correlation matrix:

	Dependent	Median_1	Median_1_PearsonCorrelation	Median_1_ZScore	Mode_1	Mode_1_PearsonCorrelation
Dependent	1.0000000000	-8.563735e-02	0.1711411470	-4.308808e-02	-0.1340054012	0.155470769
Median_1	-0.0856373506	1.000000e+00	-0.1182918368	2.063152e-02	0.9415071312	-0.065108953
Median_1_PearsonCorrelation	0.1711411470	-1.182918e-01	1.0000000000	-2.254608e-02	-0.1494335326	0.878303732
Median_1_ZScore	-0.0430880752	2.063152e-02	-0.0225460823	1.000000e+00	0.0159639225	-0.009387768
Mode_1	-0.1340054012	9.415071e-01	-0.1494335326	1.596392e-02	1.0000000000	-0.094835748
Mode_1_PearsonCorrelation	0.1554707689	-6.510895e-02	0.8783037318	-9.387768e-03	-0.0948357478	1.000000000
Mode_1_ZScore	0.0201528669	-2.413032e-02	0.0379339075	1.418864e-02	-0.0278691046	0.038040975
TrimmedMean_1	-0.0888804148	9.986275e-01	-0.1188407685	2.060127e-02	0.9471818995	-0.066819086
TrimmedMean_1_PearsonCorrelation	0.1739399769	-1.274799e-01	0.9989262327	-2.171254e-02	-0.1575204097	0.886293921
TrimmedMean_1_ZScore	-0.0238565239	-1.588355e-02	0.0160883758	7.884733e-01	-0.0218620199	0.040640650
Max_1	-0.1183847854	9.705919e-01	-0.1352572182	2.419055e-02	0.9333071100	-0.068072656
Max_1_PearsonCorrelation	0.1555681936	-2.033865e-01	0.9663965465	-2.986773e-02	-0.2129056141	0.794349096
Max_1_ZScore	-0.0085521756	2.029517e-02	-0.0185578724	-5.598143e-04	0.0191009912	-0.025706941
Min_1	-0.0595280334	9.511912e-01	-0.0713596805	1.420602e-02	0.8920654051	-0.045307249
Min_1_PearsonCorrelation	0.1793761462	-9.285784e-02	0.9636764508	-1.624016e-02	-0.1340970115	0.916256991
Min_1_ZScore	0.0087812411	-6.171868e-02	0.0268739540	3.986394e-03	-0.0492187278	0.030063594
Range_1	-0.0889318501	1.855552e-01	0.0865753128	1.435991e-02	0.1697734302	0.080600757

Showing 1 to 18 of 202 entries

As the Pearson correlation is a matrix object, it can be interacted with via subscripting. While the correlation matrix is extremely useful for identifying collinearity, at this stage the main point of interest is the relationships to the dependent variable only.

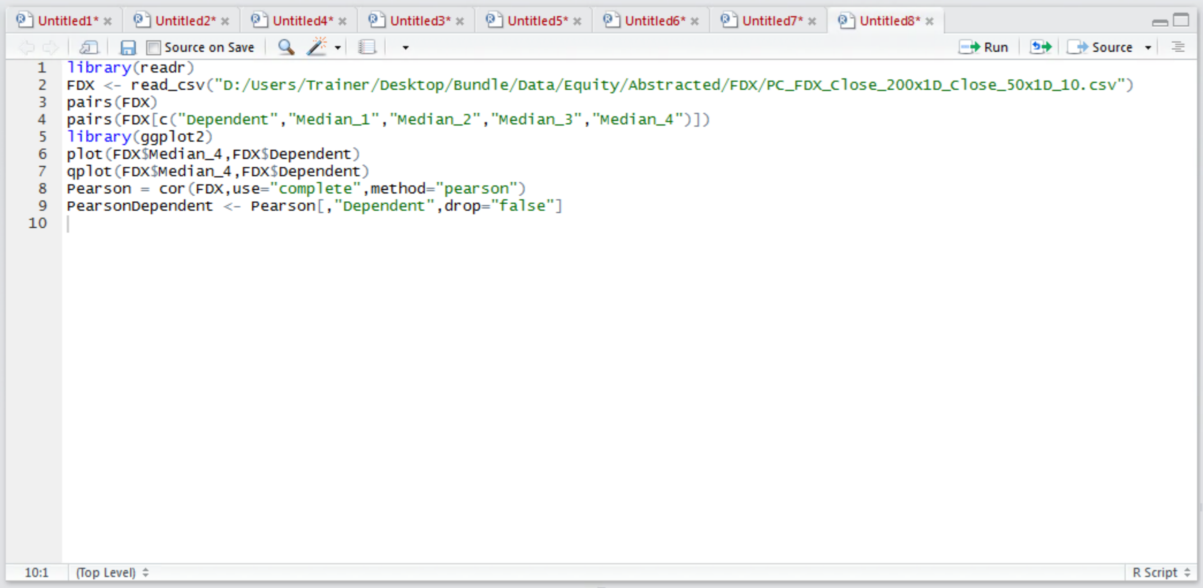
To return just the Dependent column:

PearsonDependent <- Pearson[,"Dependent",drop="false"]

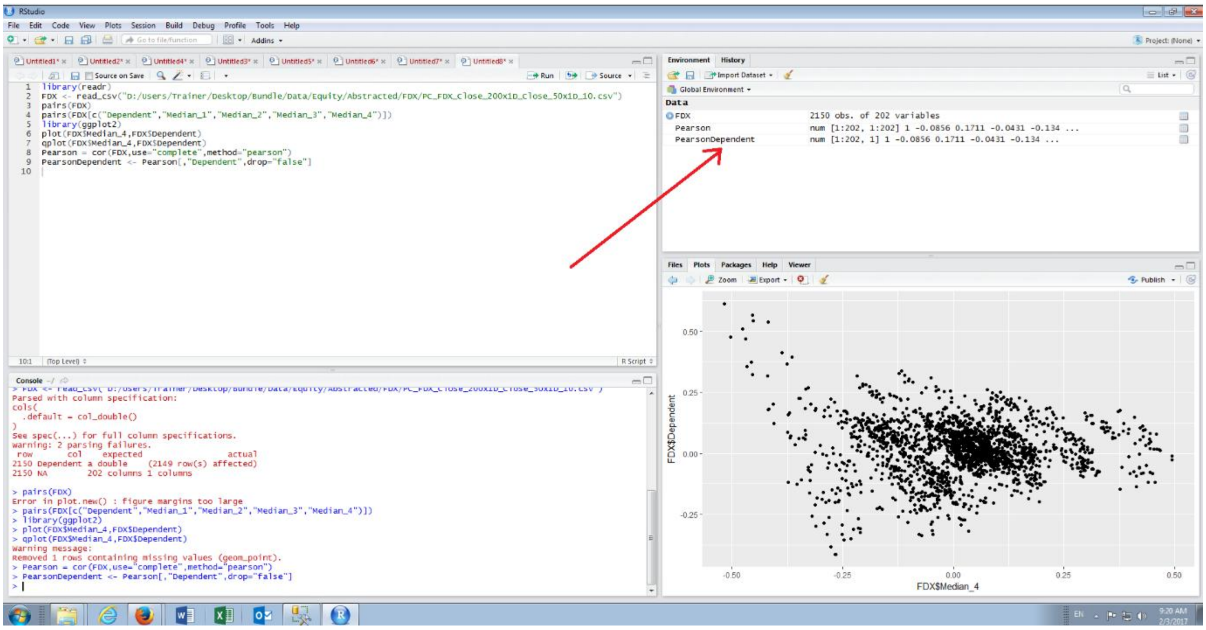
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
```

In this example the matrix is being subset to bring back all rows by leaving the first argument blank, while specifying only the "Dependent" column. By default subsetting will return the simplest structure and it cannot be assumed that it will be the same structure as the oroginal matrix, hence the drop="false" argument is used to ensure that the structure is the same (this is to say a matrix of rows and colums).

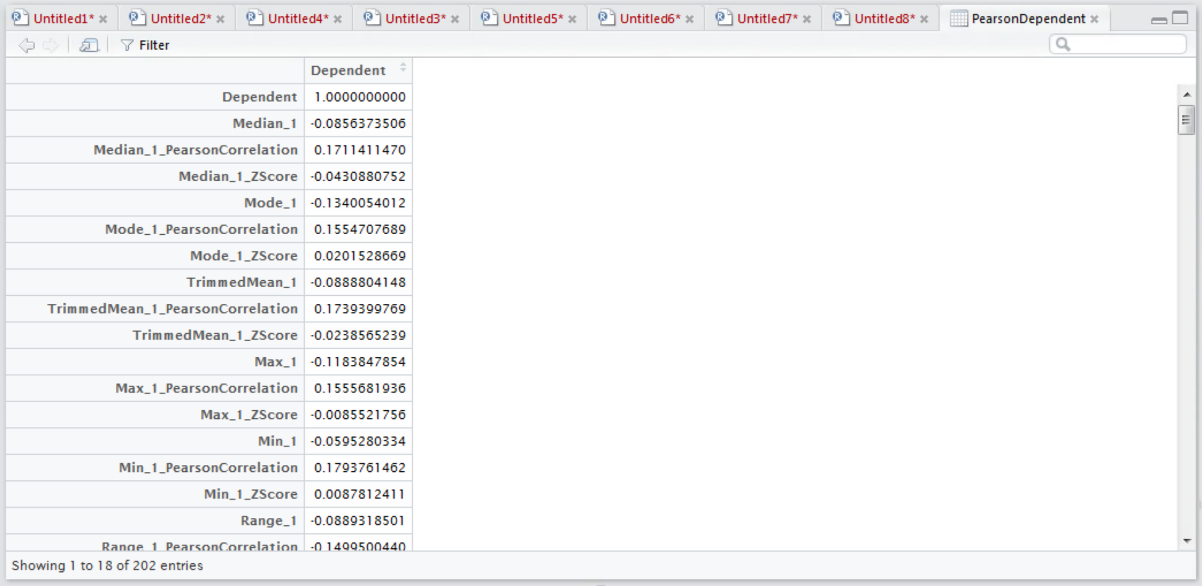
Run the line of script to console:



It can be seen that a new matrix has been created in the environment pane:



Clicking on the new matrix titled PearsonDependent will expand into the script window:



	Dependent
Dependent	1.0000000000
Median_1	-0.0856373506
Median_1_PearsonCorrelation	0.1711411470
Median_1_ZScore	-0.0430880752
Mode_1	-0.1340054012
Mode_1_PearsonCorrelation	0.1554707689
Mode_1_ZScore	0.0201528669
TrimmedMean_1	-0.0888804148
TrimmedMean_1_PearsonCorrelation	0.1739399769
TrimmedMean_1_ZScore	-0.0238565239
Max_1	-0.1183847854
Max_1_PearsonCorrelation	0.1555681936
Max_1_ZScore	-0.0085521756
Min_1	-0.0595280334
Min_1_PearsonCorrelation	0.1793761462
Min_1_ZScore	0.0087812411
Range_1	-0.0889318501
Range_1_PearsonCorrelation	-0.1499500440

Showing 1 to 18 of 202 entries

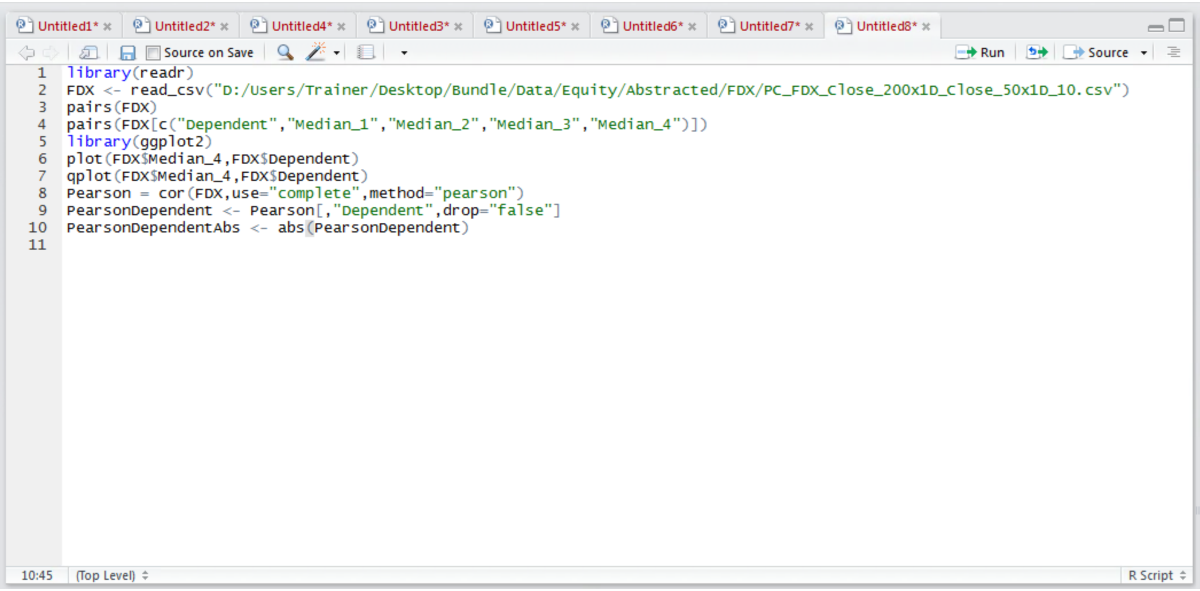
It can be seen that only the first column has been returned making the matrix less foreboding to work with in subsequent procedures.

Procedure 4: Ranking Correlation by Absolute Strength.

In procedure 86 a correlation matrix was created and the first column was transposed into a matrix by the name PearsonCorrelation. The PearsonCorrelation matrix has the strength of relationship between each of the independent variables and the dependent variables.

The first task is to order the variables by their strength of their ABSOLUTE correlation, as both -1 and +1 are equally interesting extremes. The abs() function in R makes this transformation effortless:

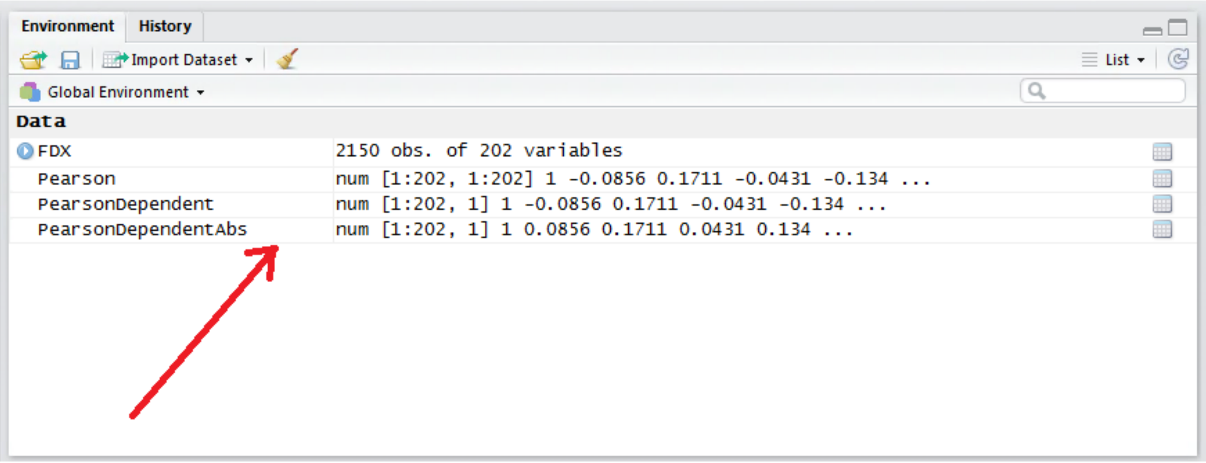
PearsonDependentAbs <- abs(PearsonDependent)



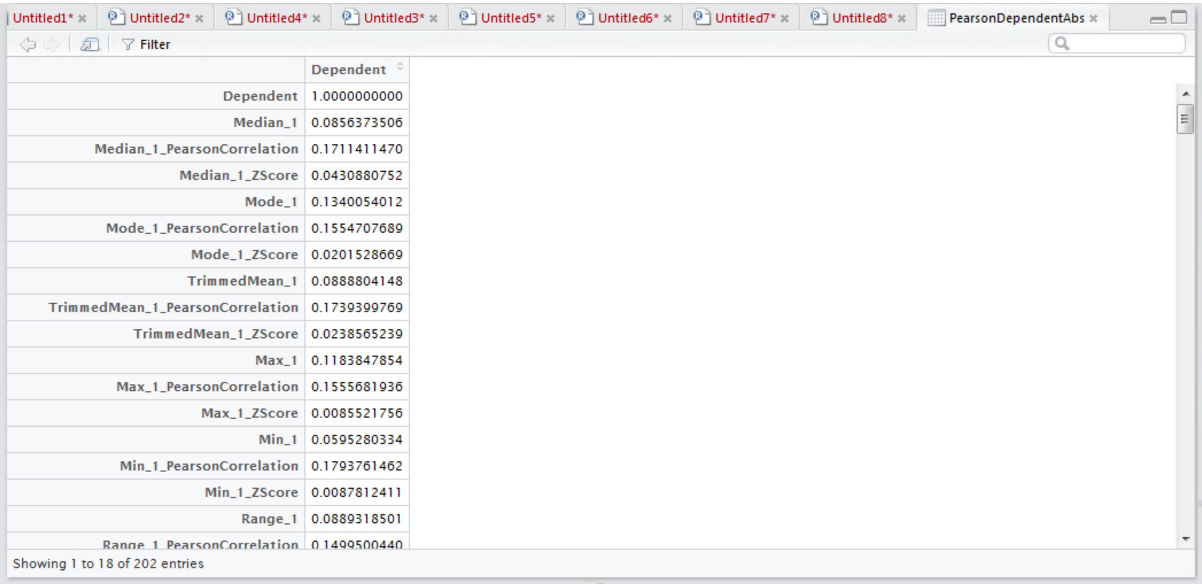
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
```

Run the line of script to console:

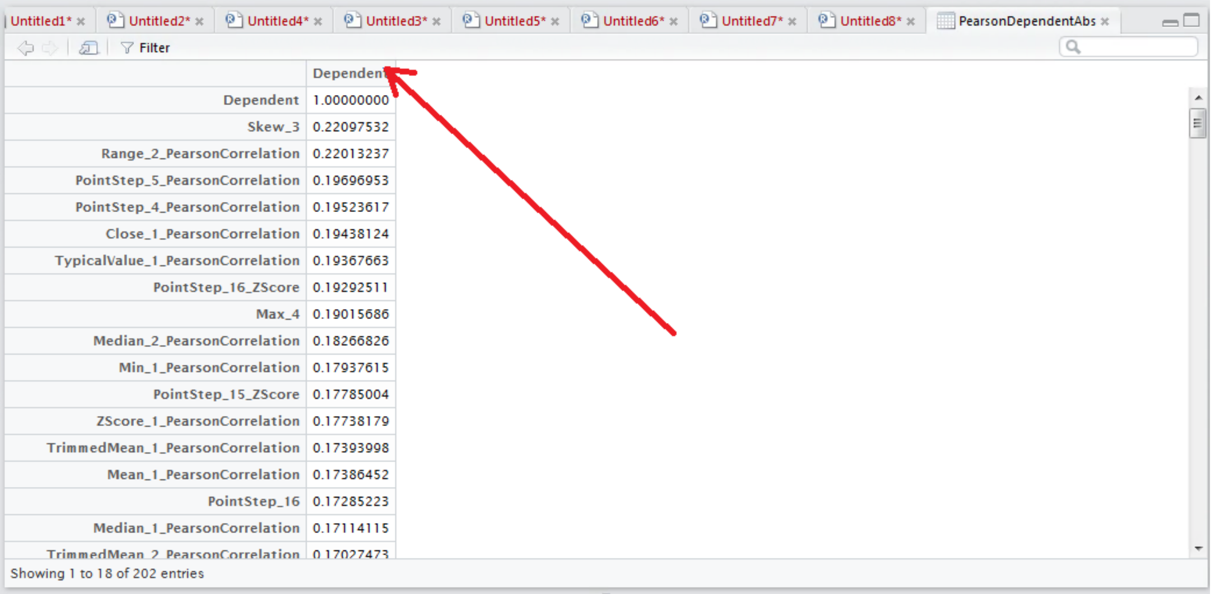
It can be seen from the environment pane window that a new matrix has been created:



In this instance, any negative number has been turned into a positive number, as observed by a single click in the environment pane:



The task remains to order the matrix by highest value to the lowest value. This can be achieved with a simple click on the column in the matrix viewer (click once for ascending, again for descending):



	Dependent
Dependent	1.00000000
Skew_3	0.22097532
Range_2_PearsonCorrelation	0.22013237
PointStep_5_PearsonCorrelation	0.19696953
PointStep_4_PearsonCorrelation	0.19523617
Close_1_PearsonCorrelation	0.19438124
TypicalValue_1_PearsonCorrelation	0.19367663
PointStep_16_ZScore	0.19292511
Max_4	0.19015686
Median_2_PearsonCorrelation	0.18266826
Min_1_PearsonCorrelation	0.17937615
PointStep_15_ZScore	0.17785004
ZScore_1_PearsonCorrelation	0.17738179
TrimmedMean_1_PearsonCorrelation	0.17393998
Mean_1_PearsonCorrelation	0.17386452
PointStep_16	0.17285223
Median_1_PearsonCorrelation	0.17114115
TrimmedMean_2_PearsonCorrelation	0.17027473

Showing 1 to 18 of 202 entries

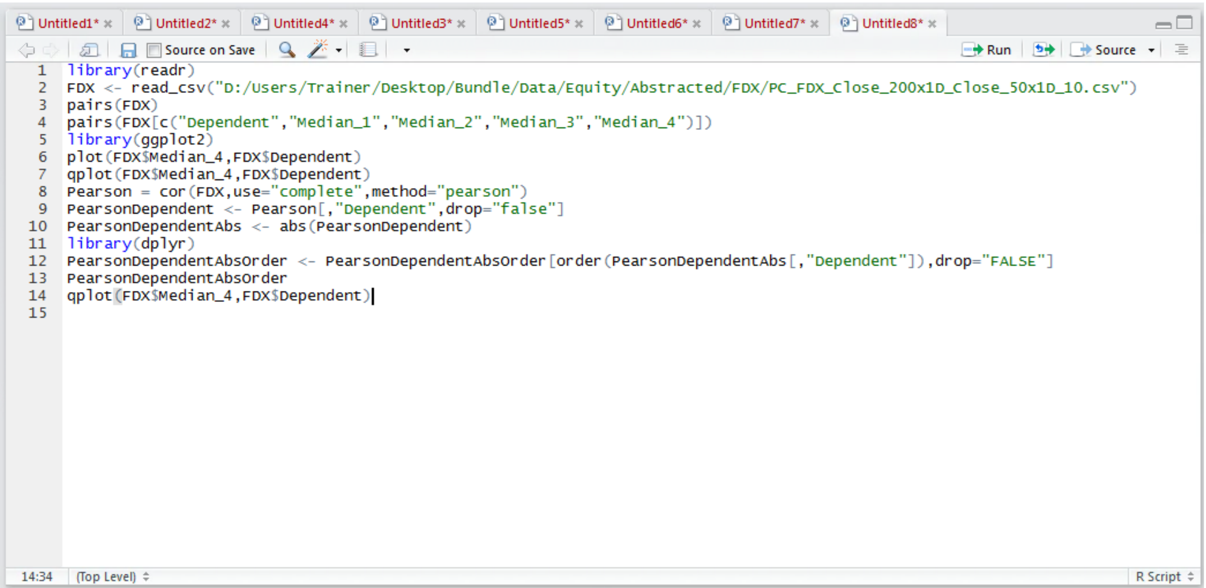
While there are methods to order a matrix in R, they are extremely convoluted and the arrange() function as presented in procedure 50 does not work, as the matrix is not a data frame.

In view of this process being exploratory and not necessarily needing to be recreated, the manual ordering in the view pane is adequate.

Procedure 5: Adding a Trend Line to a Scatter Plot.

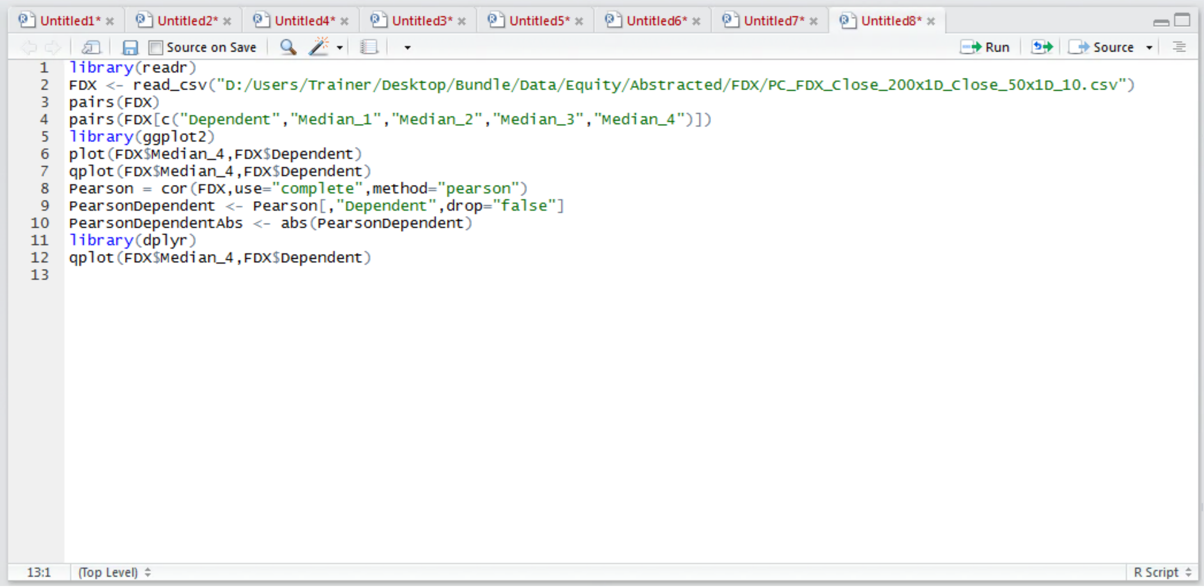
In procedure 85 a scatter plot comparing the dependent variable and the independent variable was created of Median_4. In the scatter plot, there was, just about, a relationship identified. To better visualise this relationship a trend line can be added based on a line of best fit through the points on the scatter plot.

Firstly, revisit procedure 85 to create the scatter plot using ggplot2 and the qplot() function:



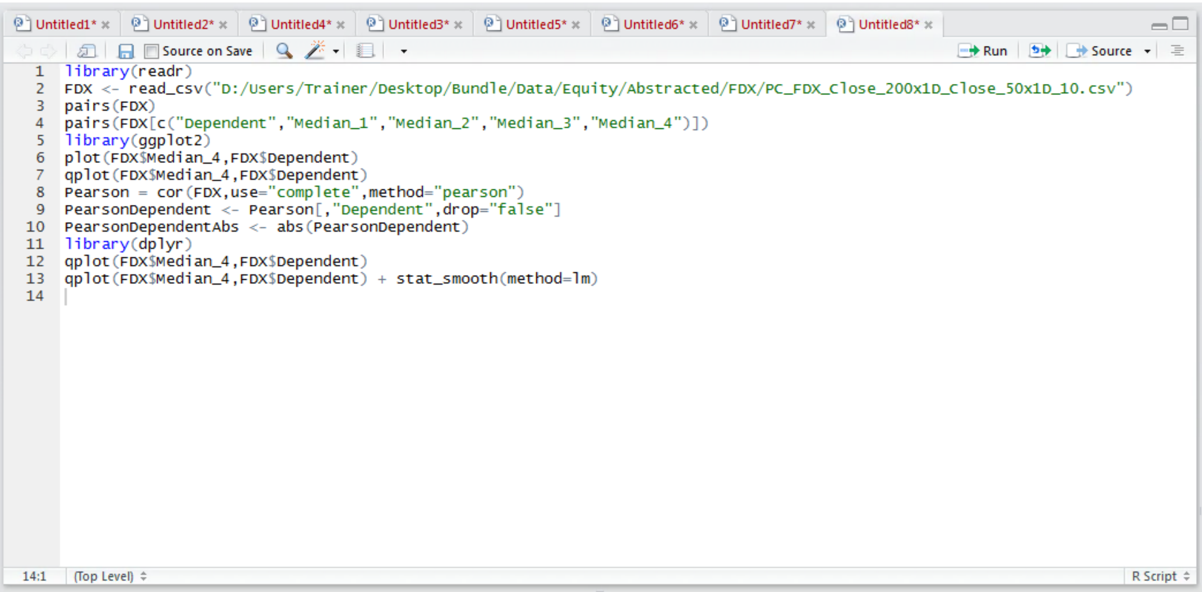
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
PearsonDependentAbsOrder <- PearsonDependentAbsOrder[order(PearsonDependentAbs[,"Dependent"]),drop="FALSE"]
PearsonDependentAbsOrder
qplot(FDX$Median_4,FDX$Dependent)
```

Run the line of script to console:



The actual formula for linear regression, as created by the lm() function is to be explained in more depth in subsequent procedures, however for the moment the lm() function is going to specified as the method of the stat_smooth() method of ggplot2:

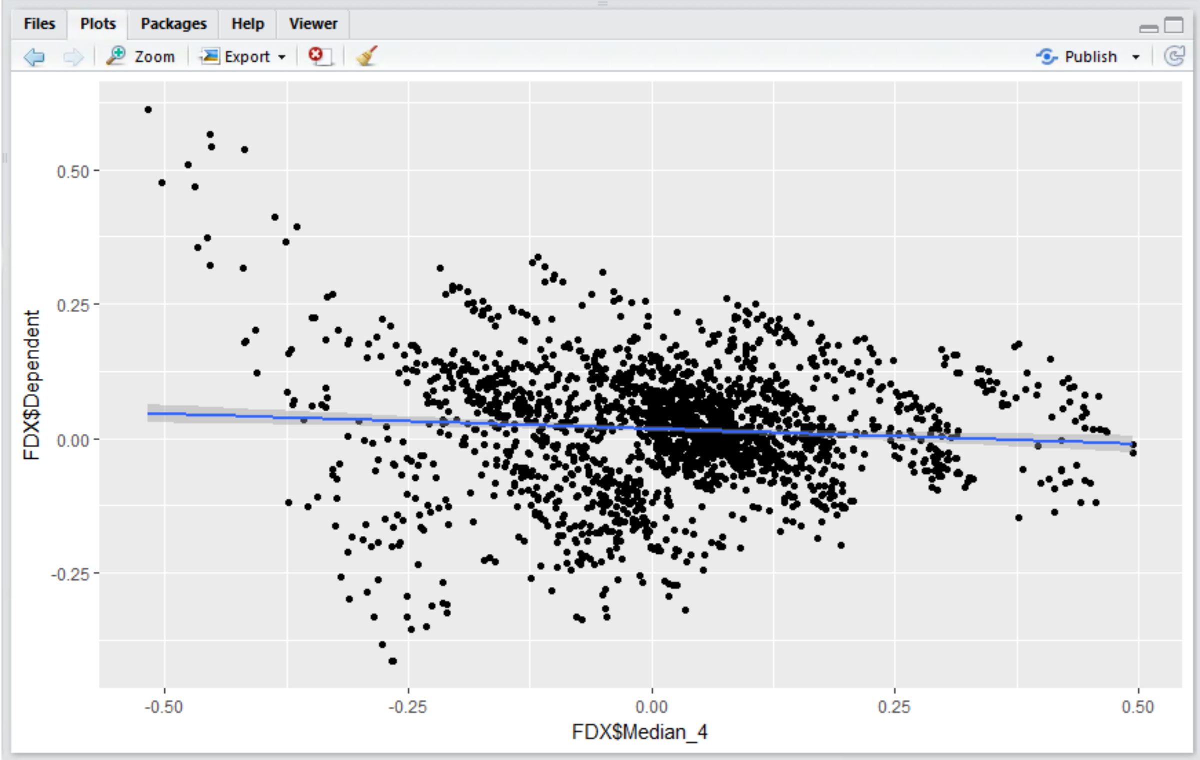
qplot(FDX$Median_4,FDX$Dependent) s



Run the line of script to console:

```
Console ~/
> PearsonDependent <- Pearson[, Dependent ,drop= false ]
> PearsonDependentAbs <- abs(PearsonDependent)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> qplot(FDX$Median_4,FDX$Dependent)
Warning message:
Removed 1 rows containing missing values (geom_point).
> qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
Warning messages:
1: Removed 1 rows containing non-finite values (stat_smooth).
2: Removed 1 rows containing missing values (geom_point).
> |
```

It can be seen that a plot has been created as in procedure 85, yet this time with a trend line representing a linear regression model:



It can be seen that there is a very shallow downward trend and this linear regression solution has some predictive power, albeit very weak in isolation (hence the importance of multiple linear regression as specified in procedure 93).

Procedure 6: Creating a One Way Linear Regression Model.

In procedure 88 the lm() function was used inside the stat_smooth() function of ggplo2 to create a linear regression solution, rather line of best fit. Naturally the lm() function can also be used to create linear regression model which can be deployed as a predictive model in its own right.

To create a linear regression model with one dependent variable and one independent variable:

LinearRegression <- lm(Dependent ~ Median_4,FDX)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
```

Run the line of script to console:

```
> PearsonDependentAbs <- abs(PearsonDependent)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> qplot(FDX$Median_4,FDX$Dependent)
Warning message:
Removed 1 rows containing missing values (geom_point).
> qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
Warning messages:
1: Removed 1 rows containing non-finite values (stat_smooth).
2: Removed 1 rows containing missing values (geom_point).
> LinearRegression <- lm(Dependent ~ Median_4,FDX)
>
```

Once the model has been computed it can be output:

LinearRegression

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
```

Run the line of script to console:

```
Console ~/
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> qplot(FDX$Median_4,FDX$Dependent)
Warning message:
Removed 1 rows containing missing values (geom_point).
> qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
Warning messages:
1: Removed 1 rows containing non-finite values (stat_smooth).
2: Removed 1 rows containing missing values (geom_point).
> LinearRegression <- lm(Dependent ~ Median_4,FDX)
> LinearRegression

Call:
lm(formula = Dependent ~ Median_4, data = FDX)

Coefficients:
(Intercept)     Median_4
    0.01758     -0.05596

> 
```

The most vital aspects of the solution are written out chiefly the Intercept and Coefficient for Median_4. Notably there is no statistical measures to appraise the overall worth of the solution.

The summary() function can be used to expand on the validity and performance of the model:

summary(LinearRegression)

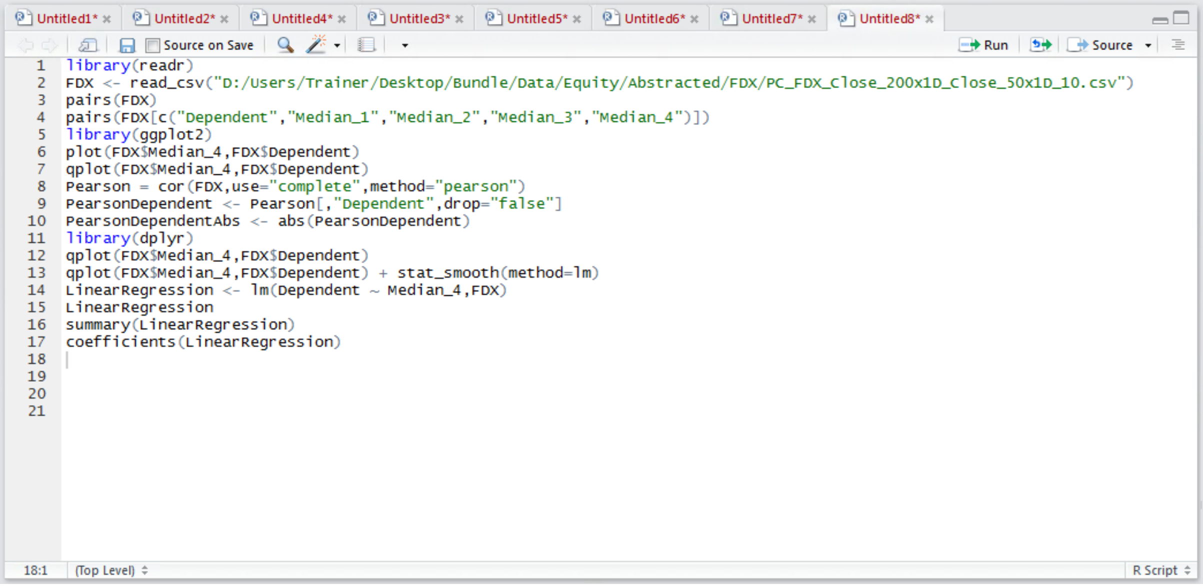
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
```

Run the line of script to console:

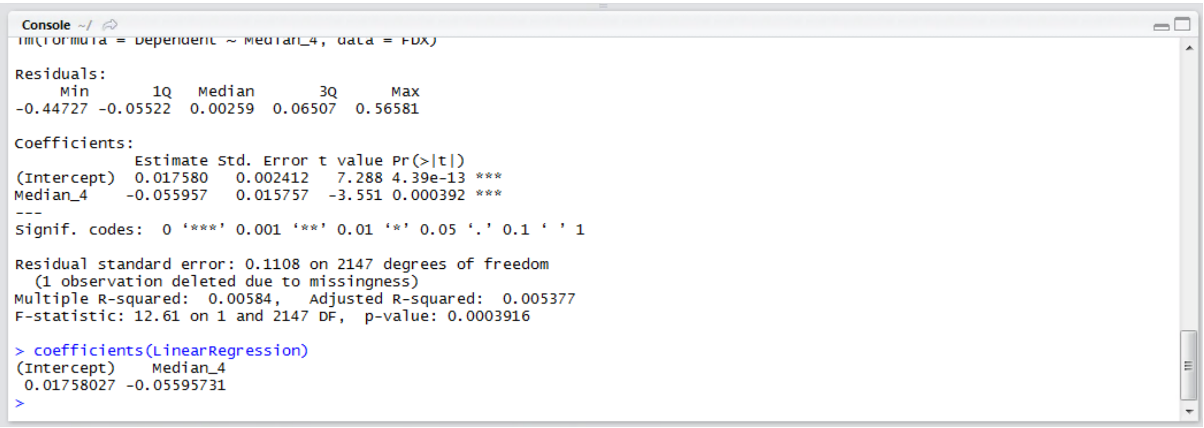
```
Console ~/
> summary(LinearRegression)

Call:
lm(formula = Dependent ~ Median_4, data = FDX)

Residuals:
     Min       1Q   Median       3Q      Max
-0.44727 -0.05522  0.00259  0.06507  0.56581

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377 
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> 
```

A more traditional Linear Regression model has now been written out. It is worth checking the precision of the coefficients to ensure that they have not been truncated, as this can lead to a profound change in the predicted values:

coeefeicents(LinearRegression)



Run the line of script to console:

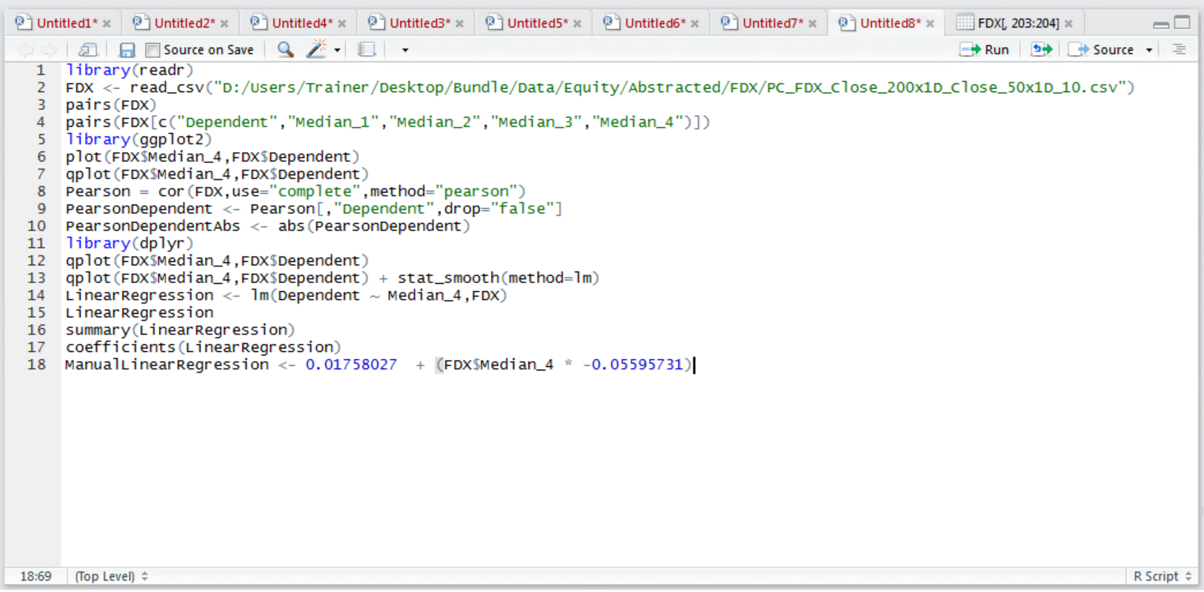


It can be seen that the coefficients written out have rather more decimal places, or precision, which will be extremely important when seeking to make accurate predictions.

Procedure 7: Deploying a One Way Linear Regression Manually with vector arithmetic.

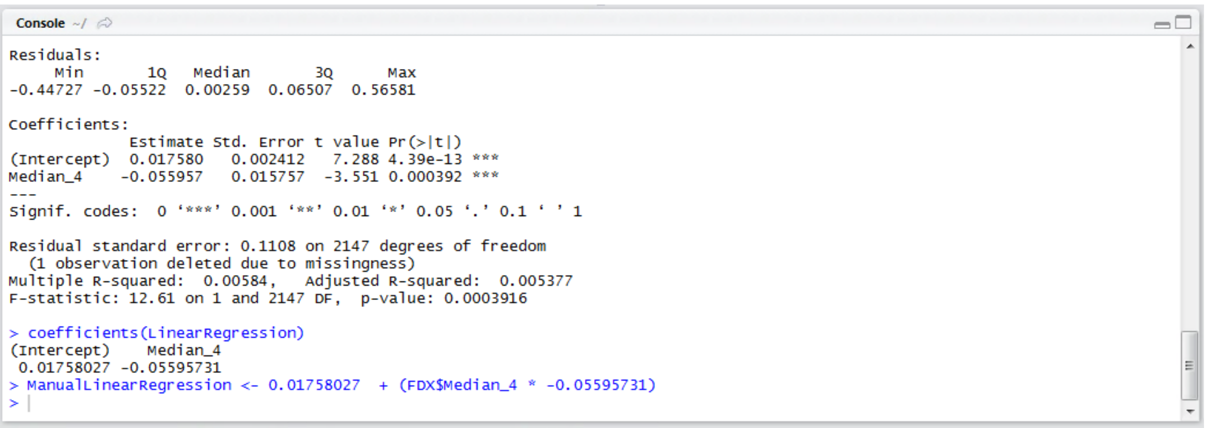
The deployment formula for a linear regression model is quite straightforward and is simply a matter of taking the intercept then adding, in this example, the Median_4 value multiplied by the coefficient:

ManualLinearRegression <- 0.01758027 + (FDX$Median_4 * -0.05595731)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
```

Run the line of script to console:



```
Residuals:
     Min       1Q   Median       3Q      Max 
-0.44727 -0.05522  0.00259  0.06507  0.56581 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377 
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4 
 0.01758027 -0.05595731 
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> 
```

As vector arithmetic, has been performed, the formula has been applied to every row of the data frame. To add this vector to the FDX data frame, procedure 53 would be executed in a similar fashion:

FDX <- mutate(FDX, ManualLinearRegression)

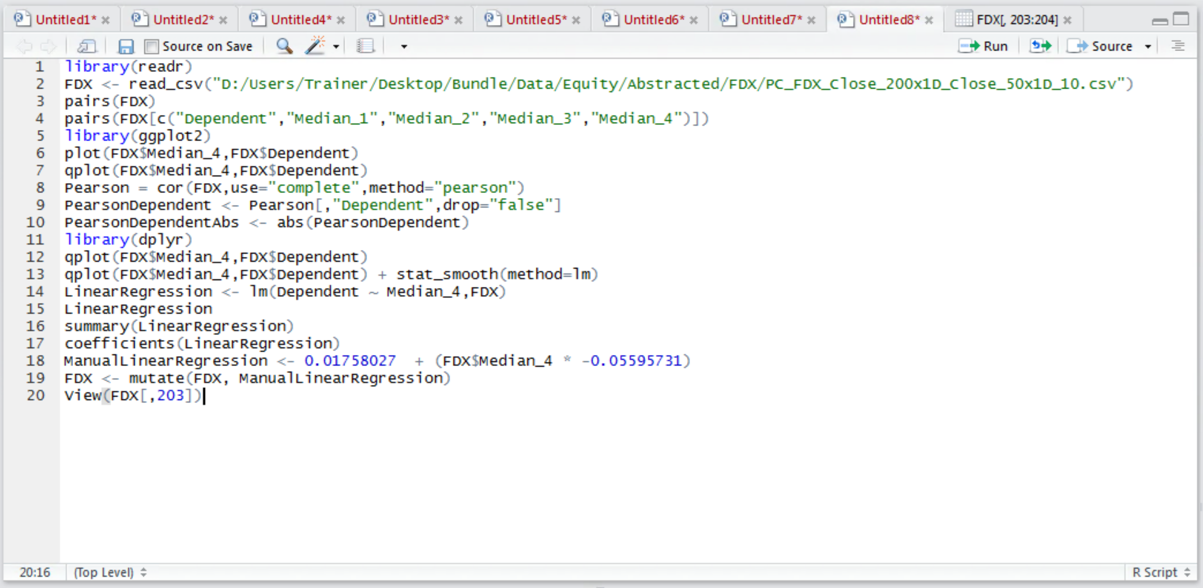
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
```

Run the line of script to console:

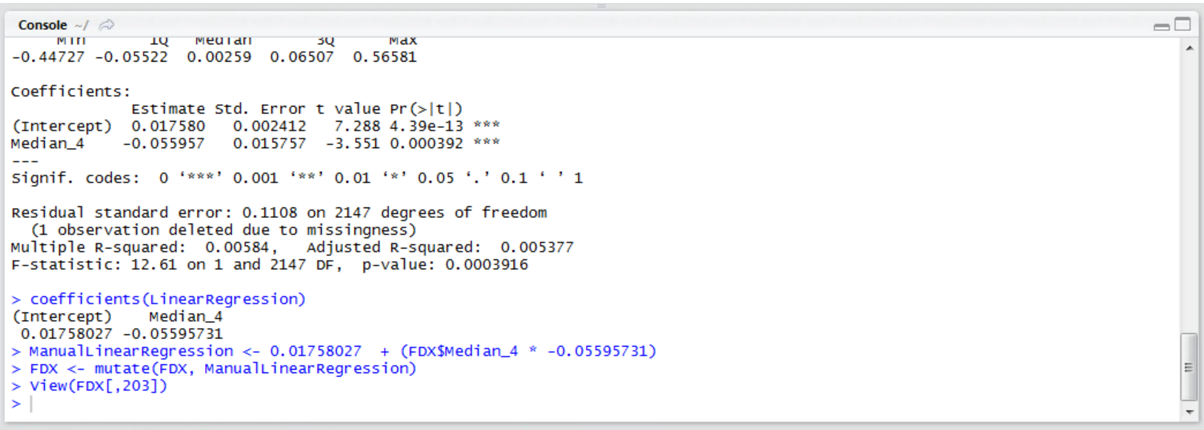
```
Residuals:
     Min       1Q   Median       3Q      Max
-0.44727 -0.05522  0.00259  0.06507  0.56581

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
>
```

The mutate() function appends the vector to the FDX data frame. To verify the column has been appended, view the FDX data frame:

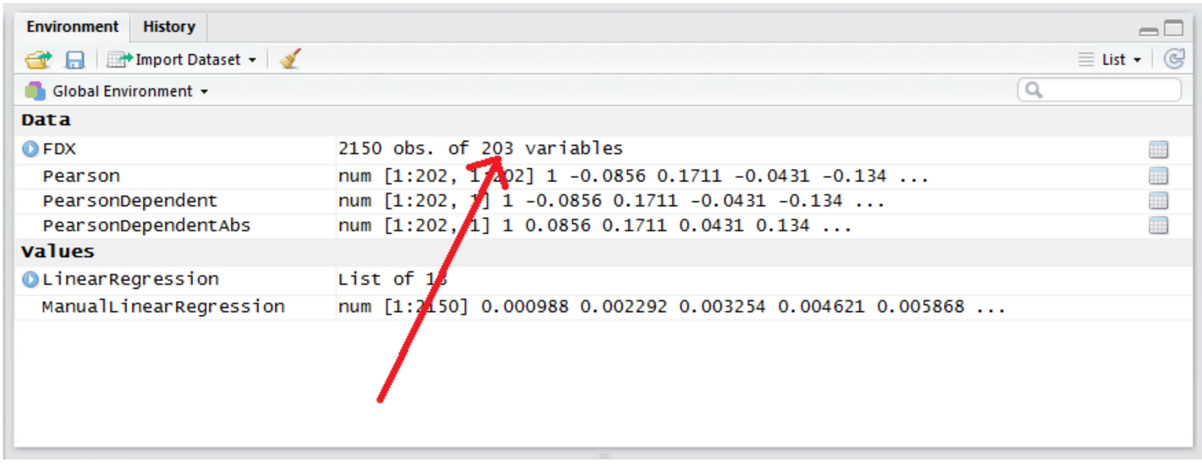
View(FDX[,203])



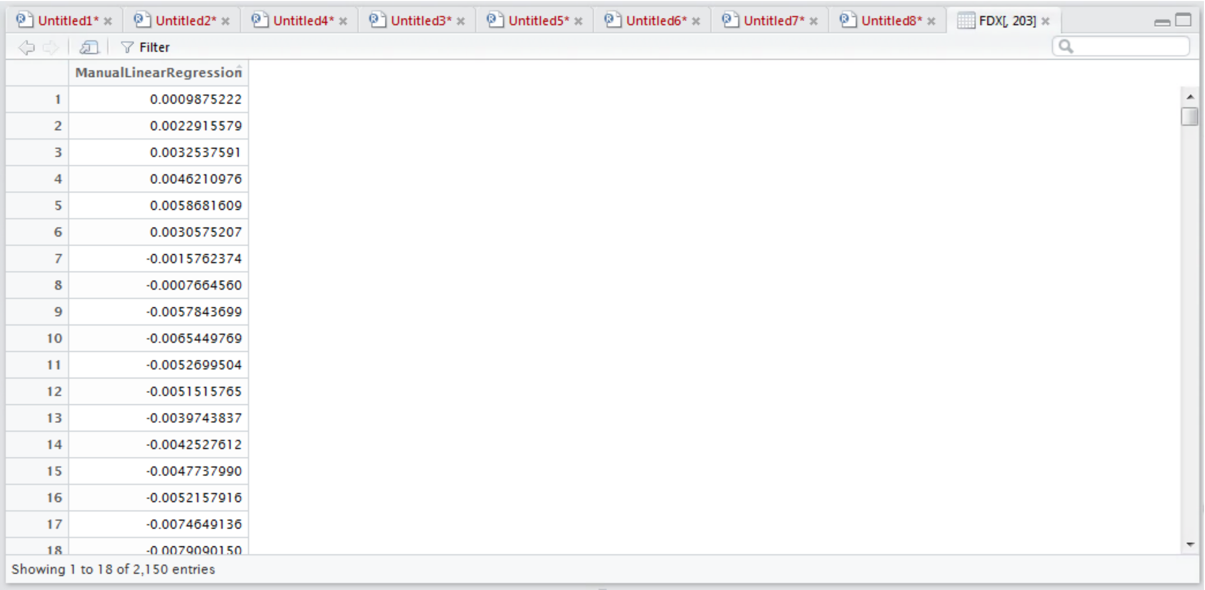
Run the line of script to console:



The use of subsetting in the call to the View() function is far less than ideal and it is to compensate for the inability of RStudio to display more than 100 columns in the grid. In this example, prior to calling the mutate() function there were 202 columns, after which there were 203:



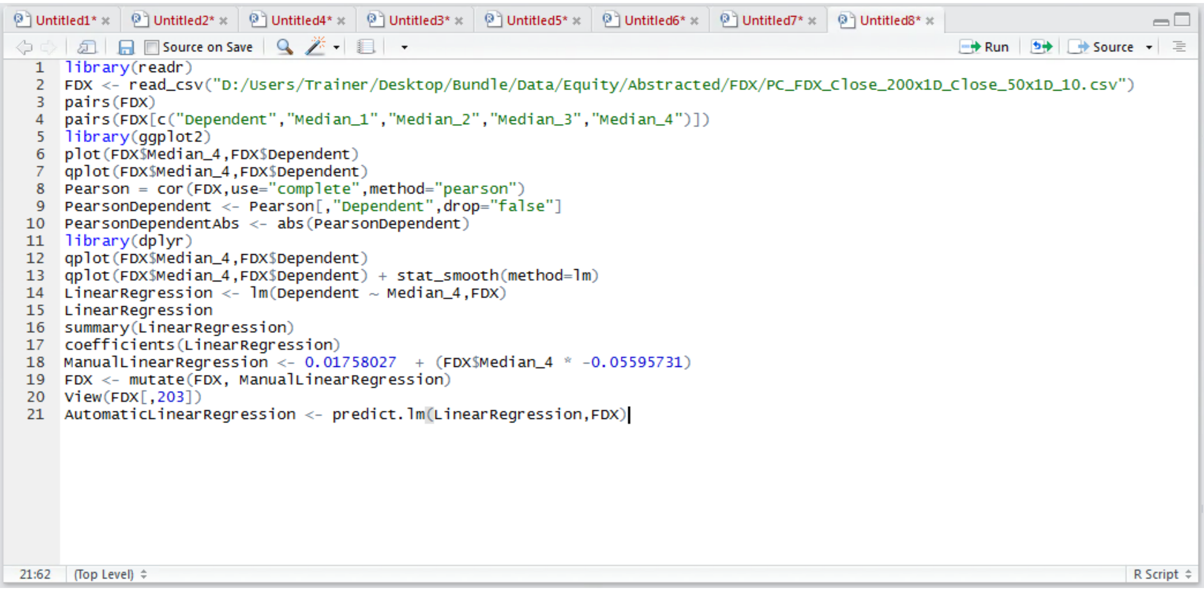
The call to the View() function in this manner yields evidence that column has been successfully added:



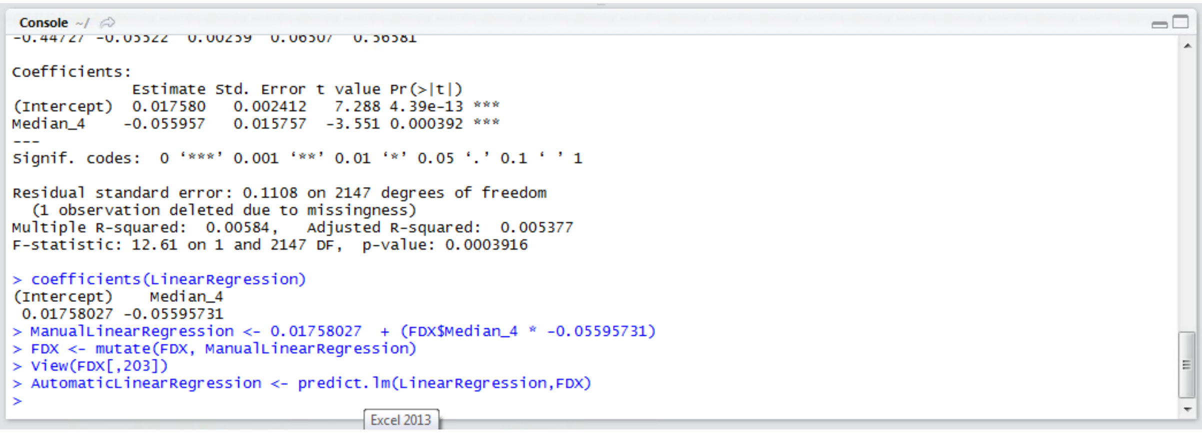
Procedure 8: Using the predict function for a one way linear regression one.

Deploying a linear regression model manually is rather simple, however, there is an even simpler method available in calling the predict() function which takes a model and a data frame as its parameter, returning a prediction vector.

AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)



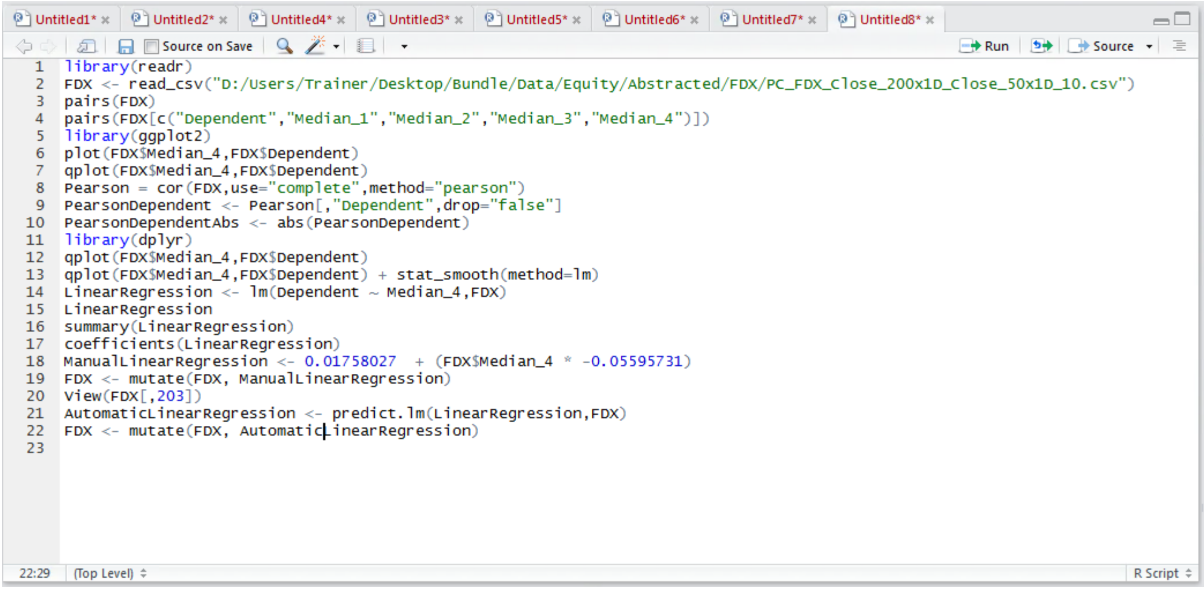
Run the line of script to console:



```
Console ~/
-0.44727 -0.05522  0.00259  0.06507  0.56581

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
> View(FDX[,203])
> AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
>
```

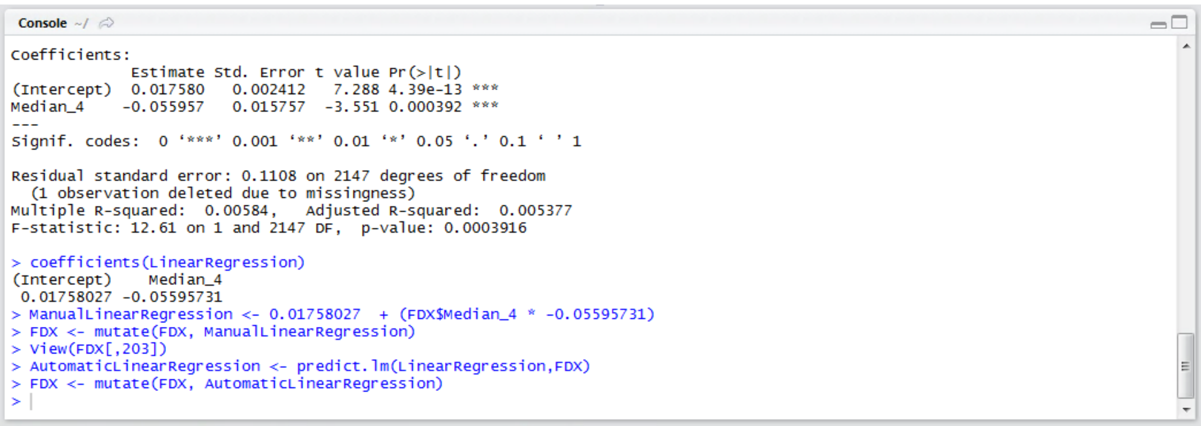
Add the newly created vector to the FDX data frame:

FDX <- mutate(FDX, AutomaticlLinearRegression)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
```

Run the line of script to console:



```
Console ~/
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
> View(FDX[,203])
> AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
> FDX <- mutate(FDX, AutomaticLinearRegression)
>
```

To view the last two columns of the data frame, containing a manually derived prediction and automatically derived prediction:

View(FDX[,203:204])

```r
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
```

Run the line of script to console:

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.017580   0.002412   7.288 4.39e-13 ***
Median_4    -0.055957   0.015757  -3.551 0.000392 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
> View(FDX[,203])
> AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
> FDX <- mutate(FDX, AutomaticLinearRegression)
> View(FDX[,203:204])
>
```

The manual and automatic prediction shown side by side are identical to each other. It follows that the automatic prediction is a much more concise means to execute the prediction based upon a linear regression model created in R:

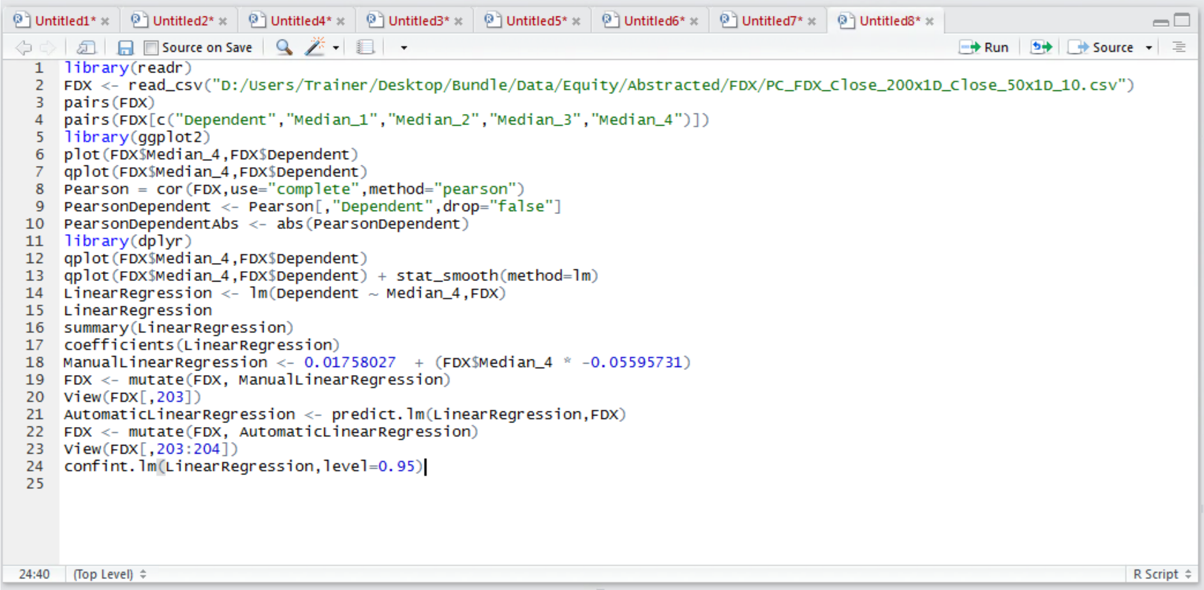
	ManualLinearRegression	AutomaticLinearRegression
1	0.01814200	0.01814200
2	0.01888378	0.01888378
3	0.01943111	0.01943111
4	0.02020890	0.02020890
5	0.02091826	0.02091826
6	0.01931948	0.01931948
7	0.01668366	0.01668366
8	0.01714429	0.01714429
9	0.01428994	0.01428994
10	0.01385728	0.01385728
11	0.01458256	0.01458256
12	0.01464989	0.01464989
13	0.01531952	0.01531952
14	0.01516117	0.01516117
15	0.01486478	0.01486478
16	0.01461336	0.01461336
17	0.01333399	0.01333399
18	0.01308137	0.01308137

Showing 1 to 18 of 2,150 entries

Procedure 9: Identifying Confidence Intervals.

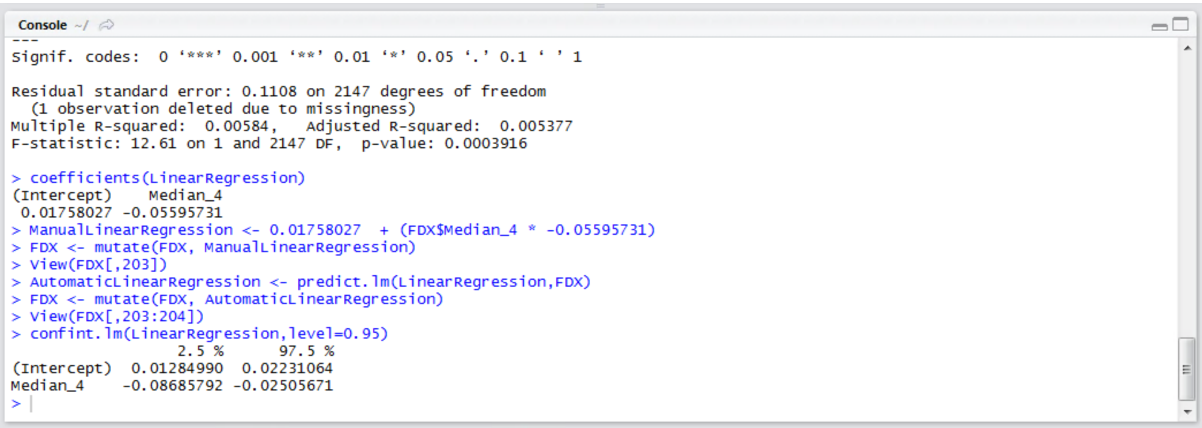
The confidence intervals can the thought of as the boundaries for which the coefficient, for a given independent variable, can be moved up and down while still maintaining statistical confidence. Unusually for regression software, the confidence intervals are not written out by default, and they need to be called by passing the linear regression model to the confint() function:

confint.lm(LinearRegression,level=0.95)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
confint.lm(LinearRegression,level=0.95)
```

Run the line of script to console:

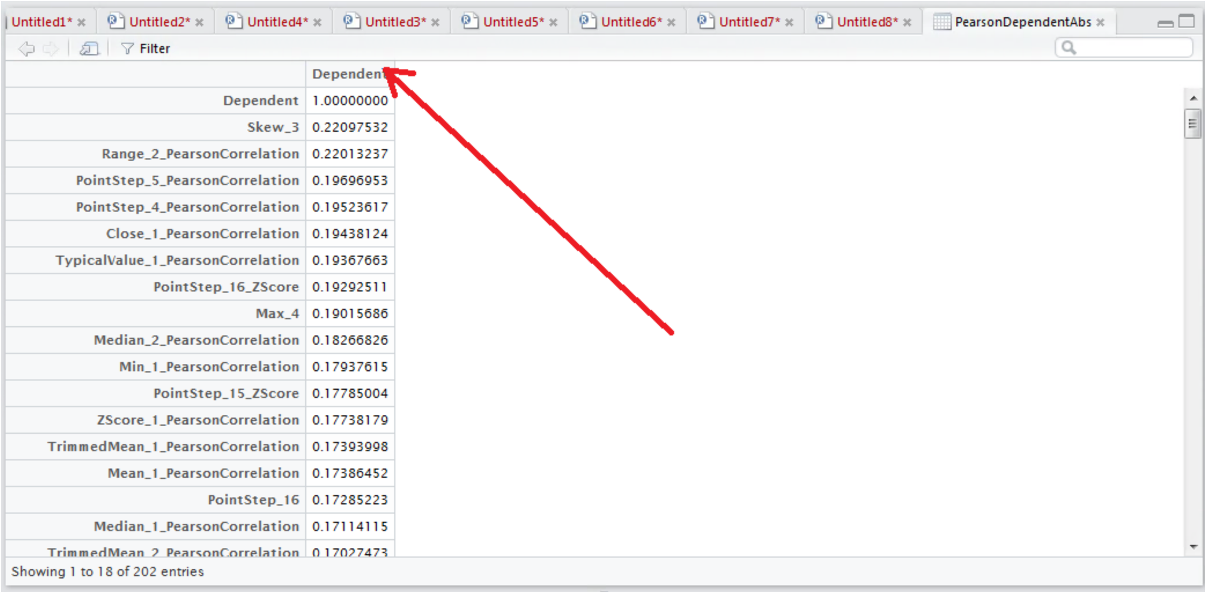


```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
> View(FDX[,203])
> AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
> FDX <- mutate(FDX, AutomaticLinearRegression)
> View(FDX[,203:204])
> confint.lm(LinearRegression,level=0.95)
                  2.5 %      97.5 %
(Intercept)  0.01284990  0.02231064
Median_4    -0.08685792 -0.02505671
>
```

The confidence intervals for each of the values required to construct the linear regression formula have been written out.

Procedure 10: Create a Stepwise Linear Regression Model.

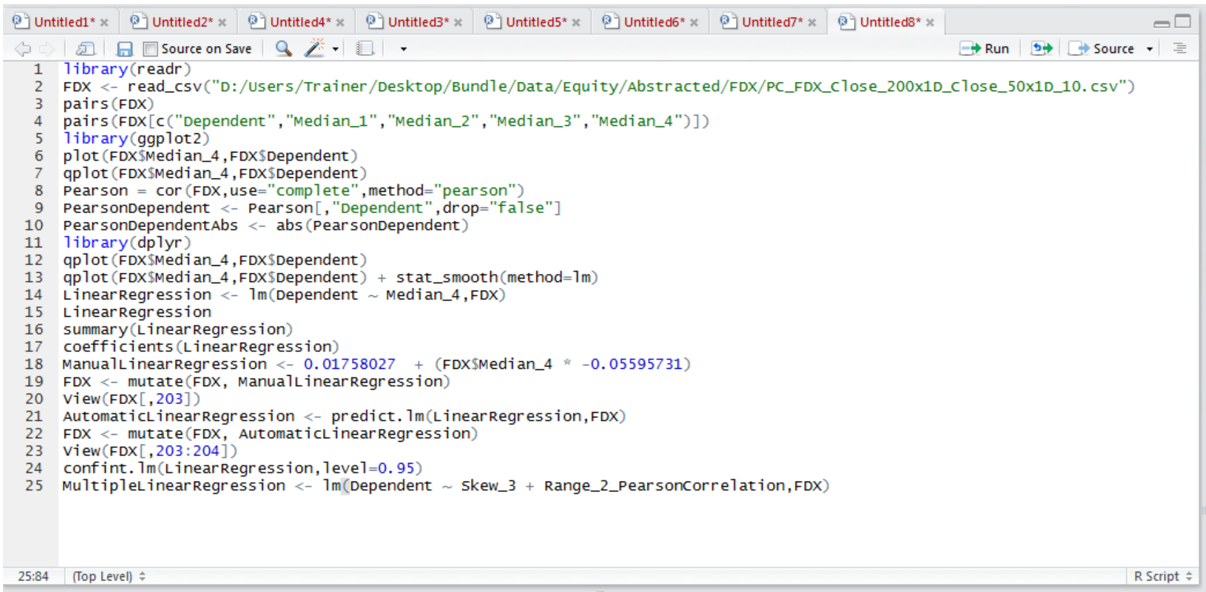
A stepwise Linear Regression model refers to adding independent variables in the order of their correlation strength in an effort to improve the overall predictive power of the model. Referring to the output of procedure 87:



	Dependent
Dependent	1.00000000
Skew_3	0.22097532
Range_2_PearsonCorrelation	0.22013237
PointStep_5_PearsonCorrelation	0.19696953
PointStep_4_PearsonCorrelation	0.19523617
Close_1_PearsonCorrelation	0.19438124
TypicalValue_1_PearsonCorrelation	0.19367663
PointStep_16_ZScore	0.19292511
Max_4	0.19015686
Median_2_PearsonCorrelation	0.18266826
Min_1_PearsonCorrelation	0.17937615
PointStep_15_ZScore	0.17785004
ZScore_1_PearsonCorrelation	0.17738179
TrimmedMean_1_PearsonCorrelation	0.17393998
Mean_1_PearsonCorrelation	0.17386452
PointStep_16	0.17285223
Median_1_PearsonCorrelation	0.17114115
TrimmedMean_2_PearsonCorrelation	0.17027473

It can be seen that the next strongest independent variable, when taking a Pearson correlation is Skew_3 followed by Range_2_Pearson_Correlation. The process of forward stepwise linear regression would be adding these variables to the model one by one, seeking improvement in the multiple r while retaining good P values. To create a multiple linear regression model of the strongest correlating independent variables:

MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation)




```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
confint.lm(LinearRegression,level=0.95)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation,FDX)
```


Run the line of script to console:

#

```
Console ~/
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1108 on 2147 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.00584,	Adjusted R-squared:  0.005377
F-statistic: 12.61 on 1 and 2147 DF,  p-value: 0.0003916

> coefficients(LinearRegression)
(Intercept)    Median_4
 0.01758027 -0.05595731
> ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
> FDX <- mutate(FDX, ManualLinearRegression)
> View(FDX[,203])
> AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
> FDX <- mutate(FDX, AutomaticLinearRegression)
> View(FDX[,203:204])
> confint.lm(LinearRegression,level=0.95)
                  2.5 %      97.5 %
(Intercept)  0.01284990  0.02231064
Median_4    -0.08685792 -0.02505671
> MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation,FDX)
> 
```

Write the summary out to observe the multiple R:

summary(MultipleLinearRegression)

```
Untitled1* × Untitled2* × Untitled4* × Untitled3* × Untitled5* × Untitled6* × Untitled7* × Untitled8* ×
Source on Save                                                        Run    Source
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
confint.lm(LinearRegression,level=0.95)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation,FDX)
summary(MultipleLinearRegression)

27:1   (Top Level)                                                    R Script
```

Run the line of script to console:

```
Console ~/
Call:
lm(formula = Dependent ~ Skew_3 + Range_2_PearsonCorrelation, 
    data = FDX)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.39862 -0.06131  0.00177  0.06549  0.59833 

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)    
(Intercept)                 0.018191   0.002438   7.461 1.24e-13 ***
Skew_3                      0.046991   0.004692  10.016  < 2e-16 ***
Range_2_PearsonCorrelation -0.054022   0.005417  -9.972  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.106 on 2146 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.09095,	Adjusted R-squared:  0.09011 
F-statistic: 107.4 on 2 and 2146 DF,  p-value: < 2.2e-16

> 
```

Several statistics are of interest in the multiple linear regression. The first is the p values relating to the overall model and the independent variables, each of these references scientific notation and so we can infer that it is an extremely small number far below the 0.05 cut off that is arbitrarily used.

Secondarily, the multiple R statistic is of interest, which will be the target of improvement in subsequent iterations.

The next step is to add the next strongest correlating independent variable, which is PointStep_5_PearsonCorrelation:

MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation + PointStep_5_PearsonCorrelation)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
confint.lm(LinearRegression,level=0.95)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation,FDX)
summary(MultipleLinearRegression)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation + PointStep_5_PearsonCorrelation,FDX)
summary(MultipleLinearRegression)
```

Run the line of script to console:

```
lm(formula = Dependent ~ Skew_3 + Range_2_PearsonCorrelation +
    PointStep_5_PearsonCorrelation, data = FDX)

Residuals:
     Min       1Q   Median       3Q      Max
-0.38125 -0.06608  0.00208  0.06469  0.58791

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                     0.046543   0.003723  12.501   <2e-16 ***
Skew_3                          0.047543   0.004589  10.360   <2e-16 ***
Range_2_PearsonCorrelation     -0.054009   0.005298 -10.193   <2e-16 ***
PointStep_5_PearsonCorrelation  0.074257   0.007488   9.917   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1036 on 2145 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.1308,	Adjusted R-squared:  0.1296
F-statistic: 107.6 on 3 and 2145 DF,  p-value: < 2.2e-16

>
```

In this example, it can be seen that the R squared has increased, so it can be inferred that the model has improved, while the p values are still extremely small. A more relevant value to pay attention to would be the adjusted R, which takes into account the number of independent variables and writes the multiple r accordingly, as such it is prudent to pay close attention to this value.

Repeat the procedure until such time as the improvement in multiple r plateaus or the performance of the P values decreases.

Procedure 11: Heat Map Correlation Matrix.

Multicollinearity refers to an Independent variable that while having a strong correlation to the Dependent Variable, also has an often-unhelpful correlation to another variable, with that variable also being quite well correlated to the Dependent Variable.

Multicollinearity can cause several issues, the most significant is the understatement of Independent Variable coefficients that would otherwise have a remarkable contribution to a model.

Multicollinearity is identified with the help of a Correlation Matrix, which has hitherto been used to identity the relationship between the Independent Variable and the Dependent Variable only.

From procedure 86 there exists a large correlation matrix:

Untitled1* | Untitled2* | Untitled4* | Untitled3* | Untitled5* | Untitled6* | Untitled7* | Untitled8* | Pearson

Filter

	Dependent	Median_1	Median_1_PearsonCorrelation	Median_1_ZScore	Mode_1	Mode_1_PearsonCorrelation	M
Dependent	1.0000000000	-8.563735e-02	0.1711411470	-4.308808e-02	-0.1340054012	0.155470769	
Median_1	-0.0856373506	1.000000e+00	-0.1182918368	2.063152e-02	0.9415071312	-0.065108953	
Median_1_PearsonCorrelation	0.1711411470	-1.182918e-01	1.0000000000	-2.254608e-02	-0.1494335326	0.878303732	
Median_1_ZScore	-0.0430880752	2.063152e-02	-0.0225460823	1.000000e+00	0.0159639225	-0.009387768	
Mode_1	-0.1340054012	9.415071e-01	-0.1494335326	1.596392e-02	1.0000000000	-0.094835748	
Mode_1_PearsonCorrelation	0.1554707689	-6.510895e-02	0.8783037318	-9.387768e-03	-0.0948357478	1.000000000	
Mode_1_ZScore	0.0201528669	-2.413032e-02	0.0379339075	1.418864e-02	-0.0278691046	0.038040975	
TrimmedMean_1	-0.0888804148	9.986275e-01	-0.1188407685	2.060127e-02	0.9471818995	-0.066819086	
TrimmedMean_1_PearsonCorrelation	0.1739399769	-1.274799e-01	0.9989262327	-2.171254e-02	-0.1575204097	0.886293921	
TrimmedMean_1_ZScore	-0.0238565239	-1.588355e-02	0.0160883758	7.884733e-01	-0.0218620199	0.040640650	
Max_1	-0.1183847854	9.705919e-01	-0.1352572182	2.419055e-02	0.9333071100	-0.068072656	
Max_1_PearsonCorrelation	0.1555681936	-2.033865e-01	0.9663965465	-2.986773e-02	-0.2129056141	0.794349096	
Max_1_ZScore	-0.0085521756	2.029517e-02	-0.0185578724	-5.598143e-04	0.0191009912	-0.025706941	
Min_1	-0.0595280334	9.511912e-01	-0.0713596805	1.420602e-02	0.8920654051	-0.045307249	
Min_1_PearsonCorrelation	0.1793761462	-9.285784e-02	0.9636764508	-1.624016e-02	-0.1340970115	0.916256991	
Min_1_ZScore	0.0087812411	-6.171868e-02	0.0268739540	3.986394e-03	-0.0492187278	0.030063594	
Range_1	-0.0889318501	1.855552e-01	0.0865753128	1.435991e-02	0.1697734302	0.080600757	

Showing 1 to 18 of 202 entries

The task is to use matrix logic to identify correlations which exceed 0.7 or is below -0.7 (as both extremes of +1 and – 1 are equally troubling in this example). The statement will use the or operator (i.e. |) and create a new correlation matrix:

PearsonColinearity <- Pearson <= -0.7 | Pearson >= 0.7

Untitled1* | Untitled2* | Untitled4* | Untitled3* | Untitled5* | Untitled6* | Untitled7* | Untitled8*

Source on Save | Run | Source

```r
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
pairs(FDX)
pairs(FDX[c("Dependent","Median_1","Median_2","Median_3","Median_4")])
library(ggplot2)
plot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent)
Pearson = cor(FDX,use="complete",method="pearson")
PearsonDependent <- Pearson[,"Dependent",drop="false"]
PearsonDependentAbs <- abs(PearsonDependent)
library(dplyr)
qplot(FDX$Median_4,FDX$Dependent)
qplot(FDX$Median_4,FDX$Dependent) + stat_smooth(method=lm)
LinearRegression <- lm(Dependent ~ Median_4,FDX)
LinearRegression
summary(LinearRegression)
coefficients(LinearRegression)
ManualLinearRegression <- 0.01758027  + (FDX$Median_4 * -0.05595731)
FDX <- mutate(FDX, ManualLinearRegression)
View(FDX[,203])
AutomaticLinearRegression <- predict.lm(LinearRegression,FDX)
FDX <- mutate(FDX, AutomaticLinearRegression)
View(FDX[,203:204])
confint.lm(LinearRegression,level=0.95)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation,FDX)
summary(MultipleLinearRegression)
MultipleLinearRegression <- lm(Dependent ~ Skew_3 + Range_2_PearsonCorrelation + PointStep_5_PearsonCorrelation,FDX)
summary(MultipleLinearRegression)
PearsonColinearity <- Pearson <= -0.7 | Pearson >= 0.7
```

30:1 (Top Level) R Script

Run the line of script to console:

```
Console ~/
PointStep_5_PearsonCorrelation, data = FDX)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.38125 -0.06608  0.00208  0.06469  0.58791 

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)    
(Intercept)                   0.046543   0.003723  12.501   <2e-16 ***
Skew_3                        0.047543   0.004589  10.360   <2e-16 ***
Range_2_PearsonCorrelation   -0.054009   0.005298 -10.193   <2e-16 ***
PointStep_5_PearsonCorrelation 0.074257   0.007488   9.917   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1036 on 2145 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.1308,	Adjusted R-squared:  0.1296 
F-statistic: 107.6 on 3 and 2145 DF,  p-value: < 2.2e-16

> PearsonColinearity <- Pearson <= 0.7 | Pearson >= 0.7
> 
```

It can be seen that a new matrix has been created in the environment pane:

Environment | History

Import Dataset | List

Global Environment

Data	
FDX	2150 obs. of 204 variables
Pearson	num [1:202, 1:202] 1 -0.0856 0.1711 -0.0431 -0.134 ...
PearsonColinearity	logi [1:202, 1:202] TRUE TRUE TRUE TRUE TRUE TRUE ...
PearsonDependent	num [1:202, 1] 1 -0.0856 0.1711 -0.0431 -0.134 ...
PearsonDependentAbs	num [1:202, 1] 1 0.0856 0.1711 0.0431 0.134 ...
Values	
AutomaticLinearRegression	Named num [1:2150] 0.0181 0.0189 0.0194 0.0202 0.0209 ...
LinearRegression	List of 13
ManualLinearRegression	num [1:2150] 0.0181 0.0189 0.0194 0.0202 0.0209 ...
MultipleLinearRegression	Large lm (13 elements, 572.6 Kb)

A click returns the matrix:

Untitled1* | Untitled2* | Untitled4* | Untitled3* | Untitled5* | Untitled6* | Untitled7* | Untitled8* | PearsonColinearity

Filter

	Dependent	Median_1	Median_1_PearsonCorrelation	Median_1_ZScore	Mode_1	Mode_1_PearsonCorrelation	Mode_1_ZSco
Dependent	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Median_1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
Median_1_PearsonCorrelation	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Median_1_ZScore	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Mode_1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
Mode_1_PearsonCorrelation	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Mode_1_ZScore	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
TrimmedMean_1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
TrimmedMean_1_PearsonCorrelation	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
TrimmedMean_1_ZScore	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Max_1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
Max_1_PearsonCorrelation	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Max_1_ZScore	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Min_1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
Min_1_PearsonCorrelation	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Min_1_ZScore	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Range_1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Showing 1 to 18 of 202 entries

This matrix now shows, with a TRUE statement, any variable combination which may suggest collinearity and requiring further inspection.

Module 9: Logistic Regression.

Logistic Regression is a modelling technique that can be used for classification where the dependent variable values are binary, 1 or 0 as such. The dataset that is used in this module is available under \Bundle\Data\FraudRisk\FraudRisk.csv which contains a set of debit card transactions whereby half of the dataset is a sample of fraudulent transactions, half of the dataset is a sample of legitimate transactions.

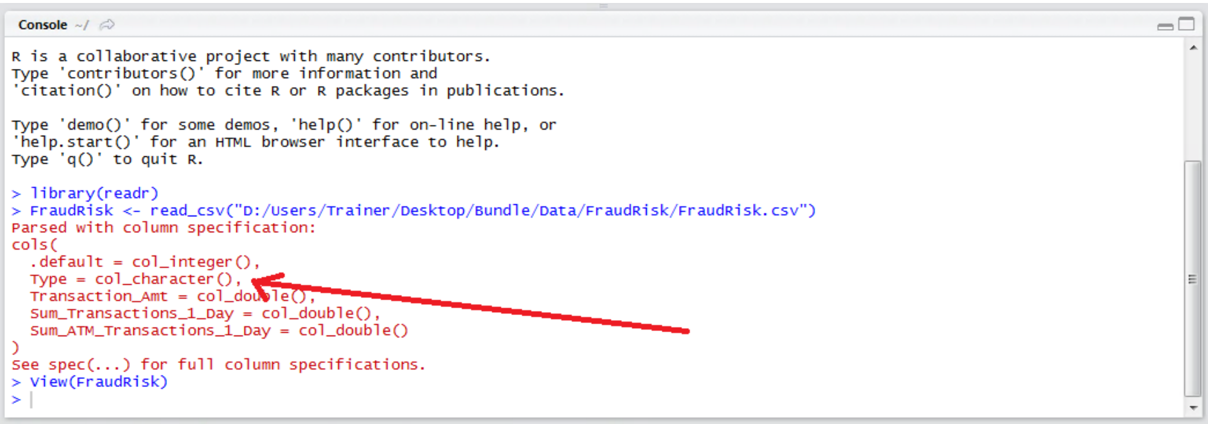
To proceed with the subsequent procedures, it is necessary to import the file FraudRisk.csv into R as per procedure 19.

Procedure 1: Pivot a Categorical Variable for Regression Analysis.

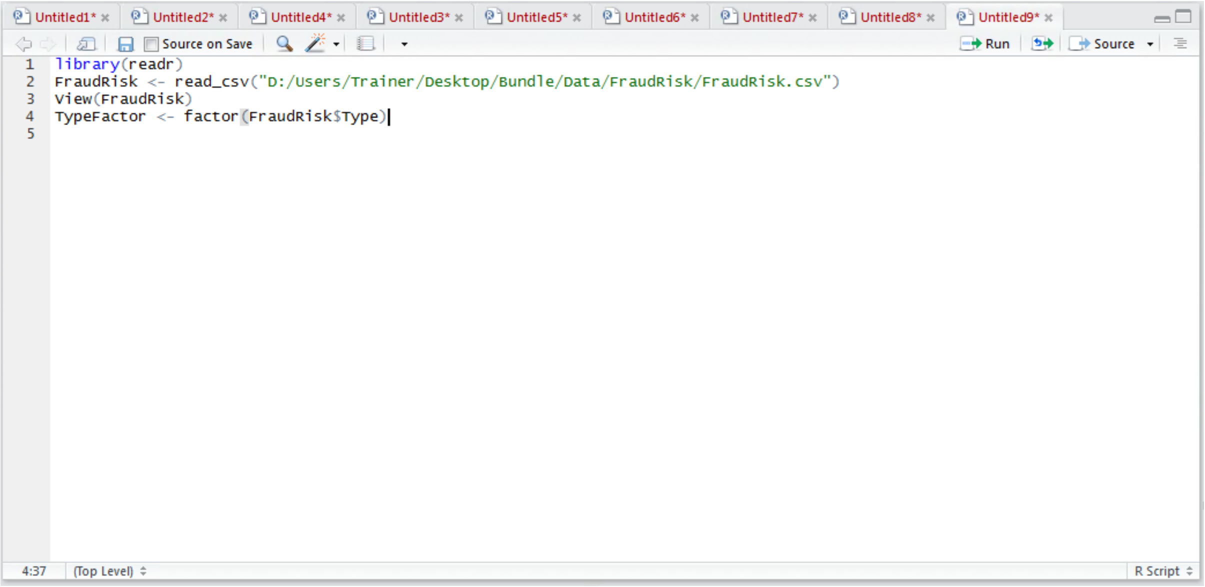
In behavioural analytics and classification, character data and numeric label data (that which has a numeric label, but obeys no standard distribution) appear quite often. It is necessary to pre-process such label data, pivoting the distinct values to their own columns, representing either a 1 or a 0, for example the transaction in this instance was either made on a Chip card (i.e. 1) or it was not (i.e. 0)

For dealing with categorical variables, and as a labour-saving tactic to avoid having to perform categorical data pivoting on each and every distinct entry in a vector, the factor functionality can be invoked and as introduced in procedure 32.

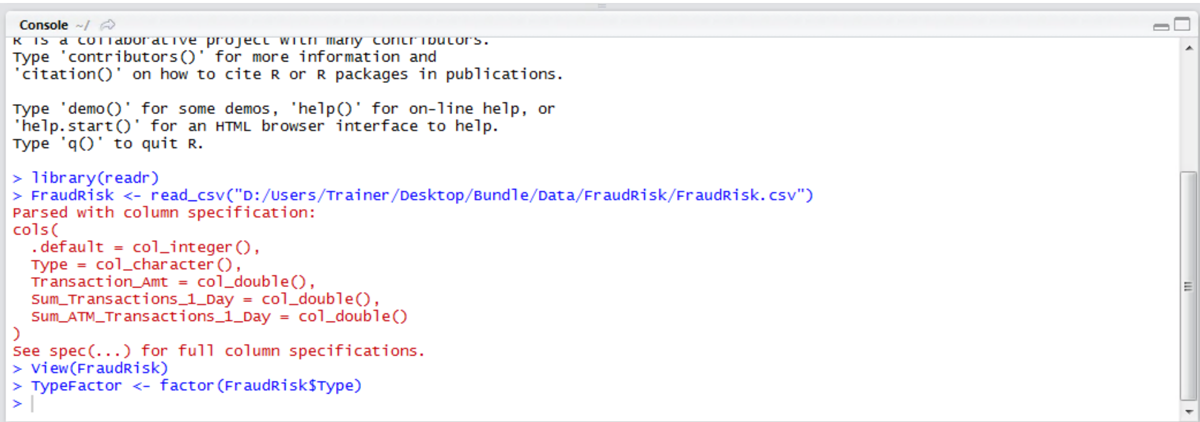
It can be seen that the data was imported with the type field taking the form of a character field:



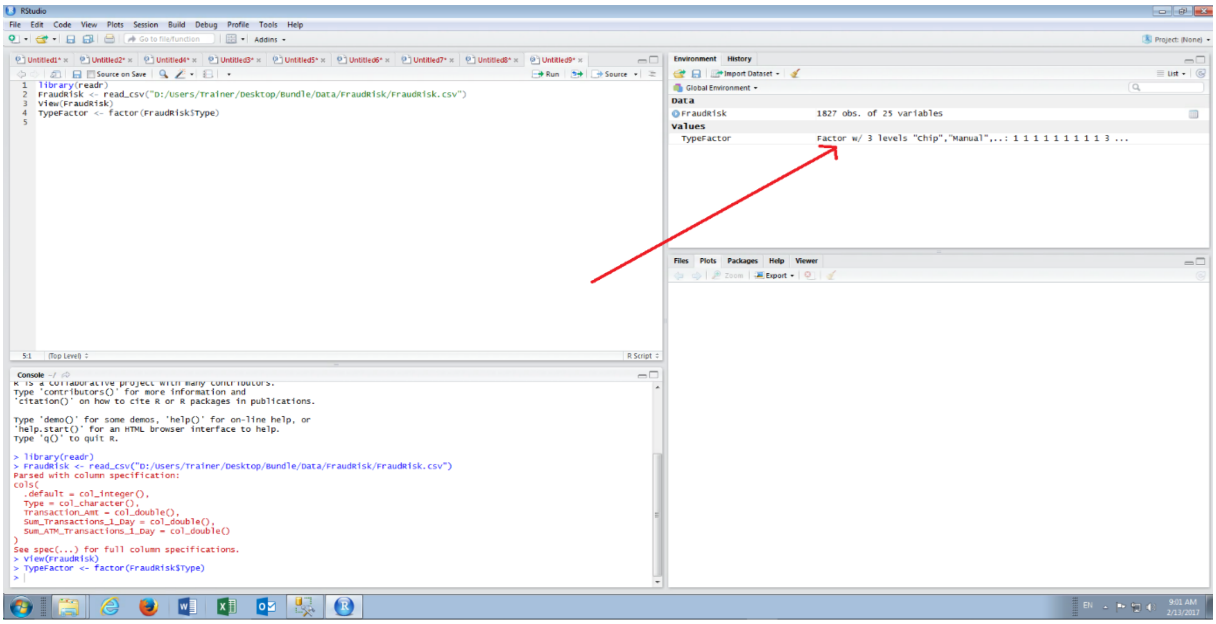
Start by creating a factor which will implicitly convert the contents of the Type column to the factor:



Run the line of script to console:



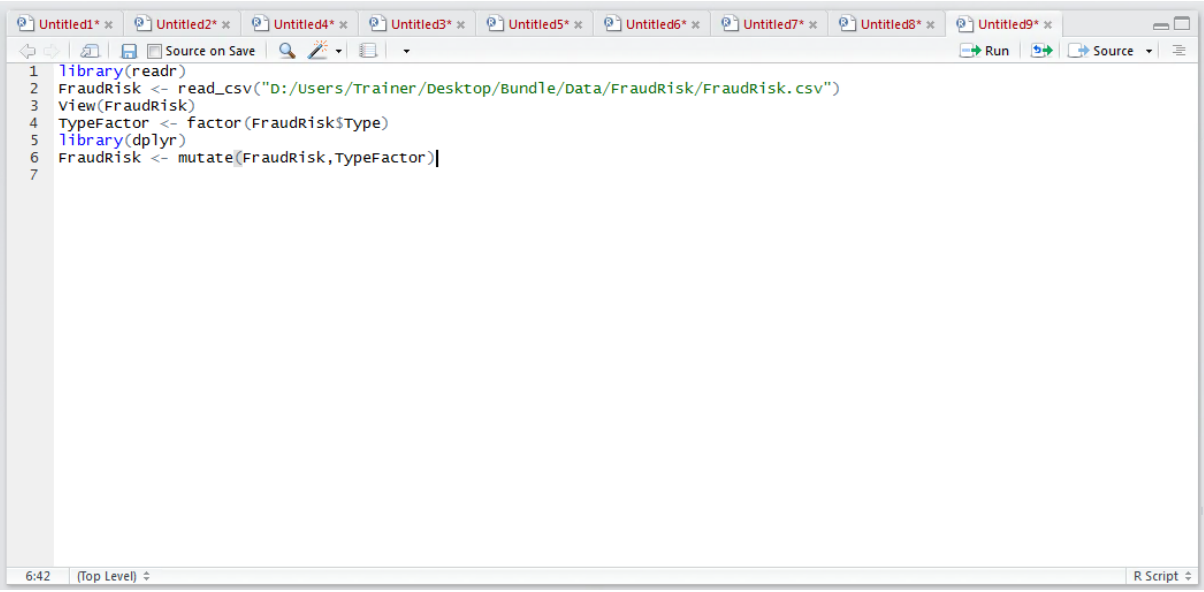
It can be seen that the factor has been created and appears in the environment pane:



All that remain is to append the newly created to factor to the FraudRisk data frame to that it can be used in subsequent analysis as procedure 52:

libarary(dplyr)

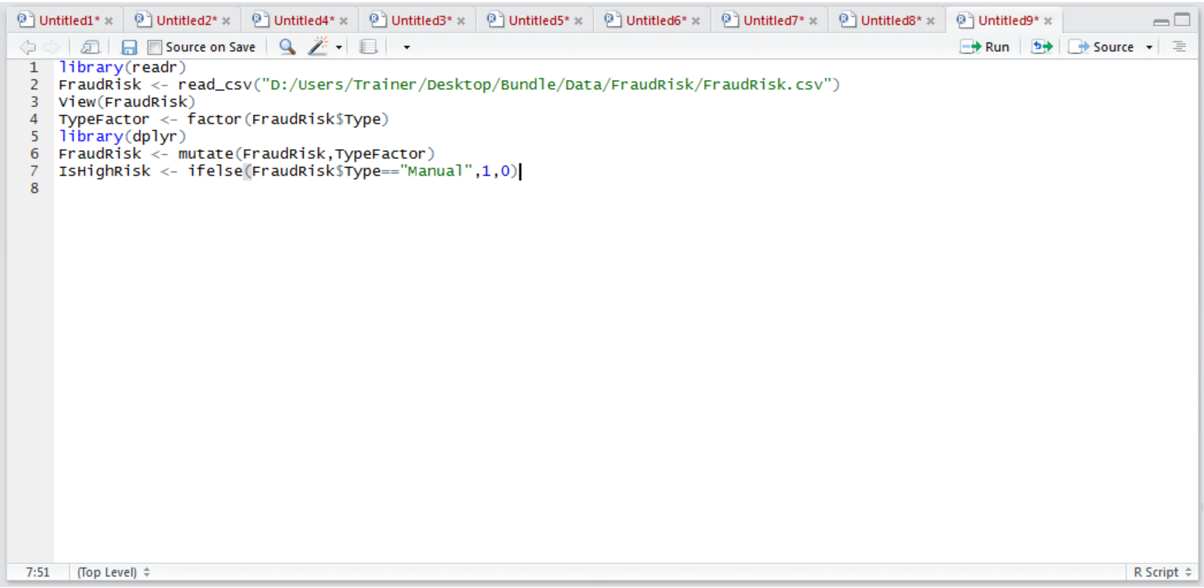
FraudRisk <- mutate(FraudRisk,TypeFactor)



Run the block of script to console:

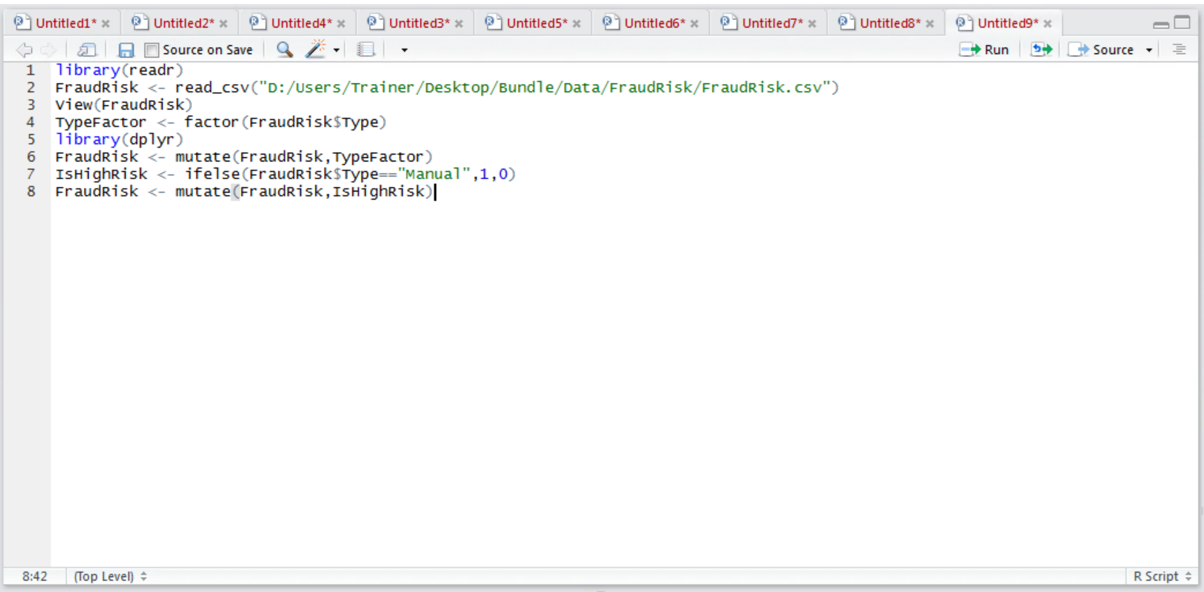
While R has a convenient data structure in the form of factors, it may well be appropriate to manually pivot data to a vector based on rudimentary if logic and \ or as part of horizontal abstraction. In this example, a vectorised comparison will be performed using the ifelse() function which will determine if a value in the Type field is equal to "Manual", in which case a the value 1 will be returned to the new vector, else 0:

IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)



Run the line of script to console:

Append the newly created vector to the FraudRisk data frame:



Run the line of script to console:

```
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> View(FraudRisk)
> TypeFactor <- factor(FraudRisk$Type)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
>
```

Procedure 2: Create an Abstraction Deviation Independent Vector.

In behavioural analytics, especially, one of the most powerful improvements that can be made to a variable is a transformation to compare the value for that records against the value typically observed in this vector for a customer \ product \ portfolio. There are of course several normalisations that are appropriate for such a task, such as a Z score, however in this instance given the data being skewed a range normalisation may be more appropriate.

A range normalisation will establish the largest value observed in the vector, the smallest value and establish where a test value exists on that range in percentage terms. In this example, a range normalisation will be performed on the columns Count_Transactions_1_Day. Firstly, establish the maximum and minimum values as similar to procedure 56:

Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)

Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)

```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
```

Run the block of script to console:

```
Console ~/
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> View(FraudRisk)
> TypeFactor <- factor(FraudRisk$Type)
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
> FraudRisk <- mutate(FraudRisk,IsHighRisk)
> Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
> 
```

At this stage, the minimum and maximum values have been stored as vectors for Count_Transactions_1_Day. To create a new vector as a range normalisation:

Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count_Transactions_1_Day - Min_Count_Transactions_1_Day)

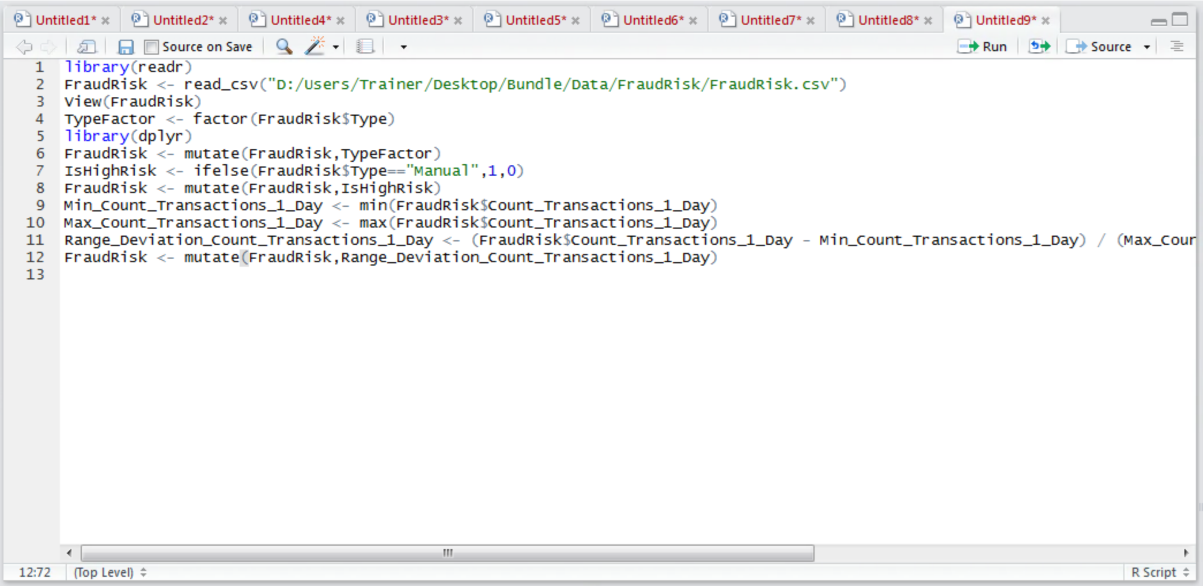
```
Untitled1* Untitled2* Untitled4* Untitled3* Untitled5* Untitled6* Untitled7* Untitled8* Untitled9*
Source on Save                                              Run   Source

le/Data/FraudRisk/FraudRisk.csv")








Transactions_1_Day)
Transactions_1_Day)
sk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count_Transactions_1_Day - Min_Count_Transactions_1_Day)

11:176  (Top Level)                                                  R Script
```

Rin the line of script to console:

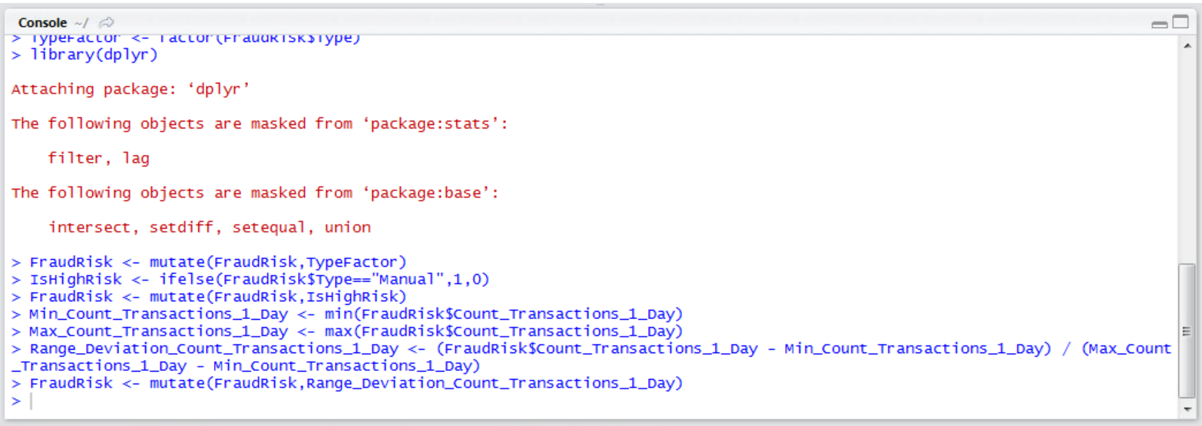
```
Console ~/
> View(FraudRisk)
> TypeFactor <- factor(FraudRisk$Type)
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
> FraudRisk <- mutate(FraudRisk,IsHighRisk)
> Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
> Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
> Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count
_Transactions_1_Day - Min_Count_Transactions_1_Day)
> 
```

Append the newly created vector to the FraudRisk data frame:

FraudRisk <- mutate(FraudRisk, Range_Deviation_Count_Transactions_1_Day)



Run the line of script to console:



Procedure 3: Fit a one-way Log Curve on a Plot.

As in procedure 88 where a relationship between two variables was appraised using a linear regression, rather ordinary least squares estimation, a similar method exists in R for appraising the extent to which two variables fit a log curve. Start by plotting the dependent variable, fraud, with the independent variable Count_Transactions_1_Day as procedure 85:

library(ggplot2)

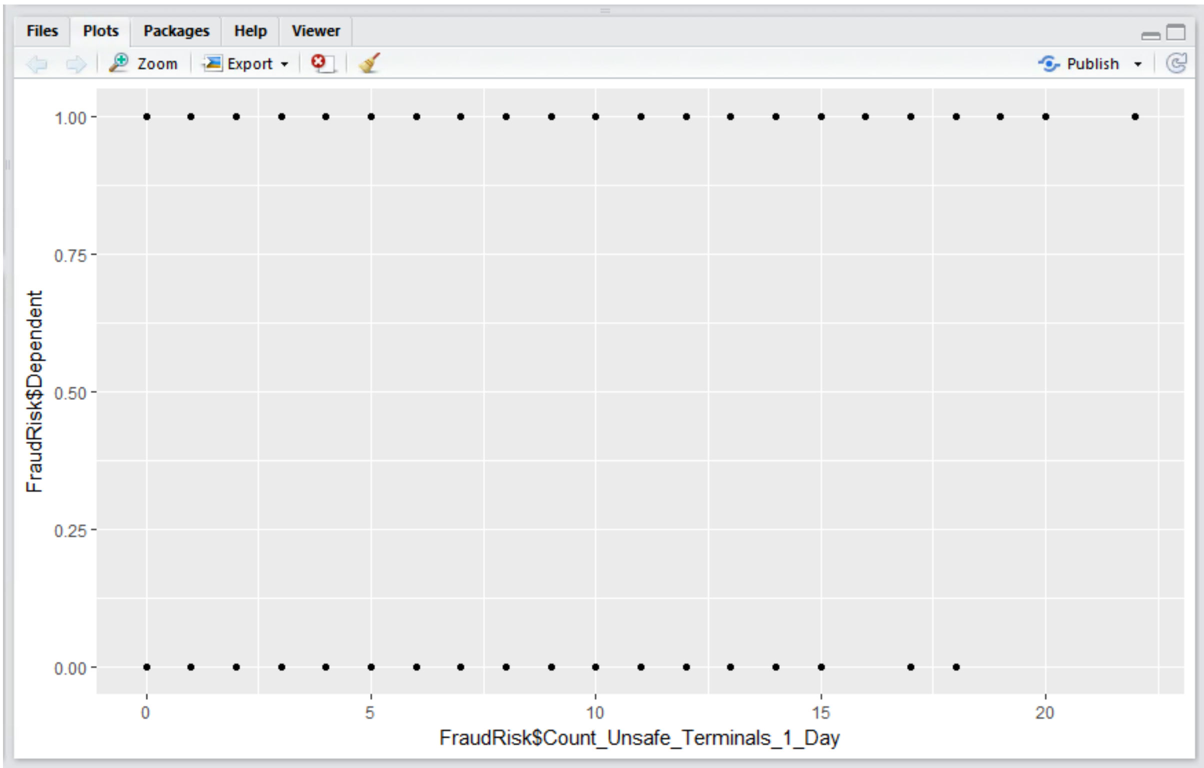
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)

```
le/Data/FraudRisk/FraudRisk.csv")







Transactions_1_Day)
Transactions_1_Day)
sk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count_Transactions_1_Day - Min_Count_Transactions_1_Day)
Transactions_1_Day)

sk$Dependent) + stat_smooth(method="glm", method.args=list(family="binomial"))
```

Run the block of script to console:

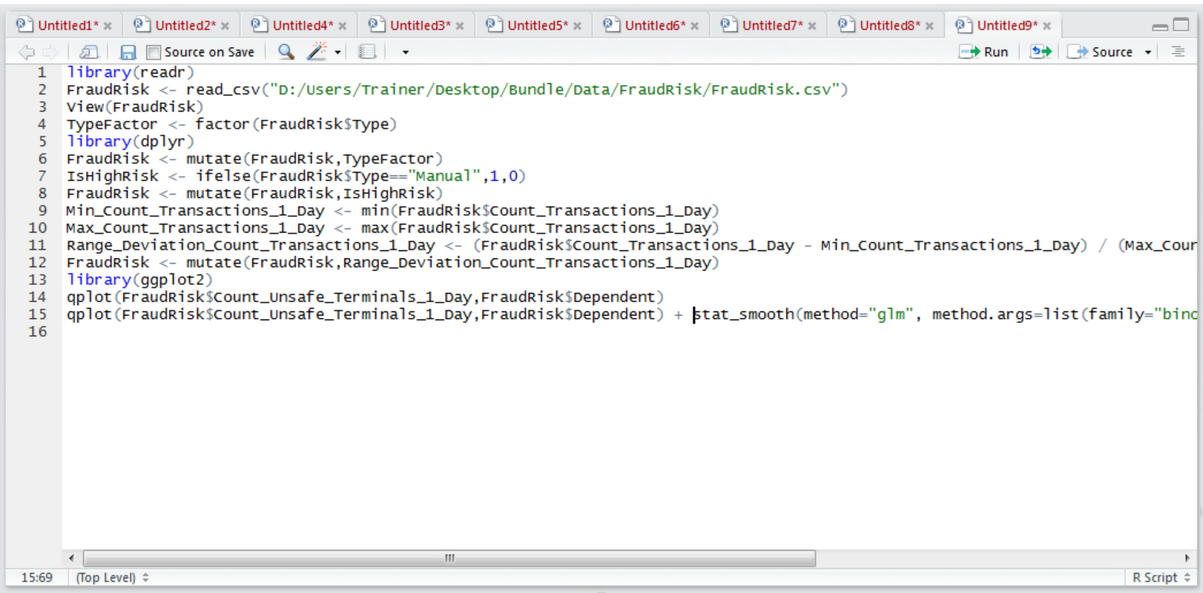
```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
> FraudRisk <- mutate(FraudRisk,IsHighRisk)
> Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
> Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
> Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count
_Transactions_1_Day - Min_Count_Transactions_1_Day)
> FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
> library(ggplot2)
> qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
>
```

It can be seen that a plot has been created between the variable Count_Unsafe_Terminals_1_Day and the Dependent variable, and on the basis, that the fraud can either be or not, it has plotted nothing between the points on the Y axis:



To estimate an appropriate logistic regression curve through the points use the glm function of the statsmooth() function:

qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binomial"))



Run the line of script to console to create the plot with a fitted logistic curve:

```
The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
> FraudRisk <- mutate(FraudRisk,IsHighRisk)
> Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
> Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
> Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count
_Transactions_1_Day - Min_Count_Transactions_1_Day)
> FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
> library(ggplot2)
> qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
> qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binom
ial"))
>
```

It can be observed that there is a defined log curve that would suggest that the more and more unsafe terminals a customer uses, the more and more certain it becomes that the account may be subject to fraud. It follows that it can be assumed that this value will have some validity for logistic regression modelling.

Procedure 4: Forward Stepwise Logistic Regression.

As procedure 97 alludes, whereas the linear regression function in R was lm(), the logistic regression function is glm(), with supplementary parameters specifying the family as being a binomial distribution (which is a stalwart distribution for classification problems). As in procedure 89 which create a linear regression model, the syntax is very similar to create a logistic regression model, albeit including the family argument:

LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")

```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
```

Run the line of script to console:

```
Console ~/
The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FraudRisk <- mutate(FraudRisk,TypeFactor)
> IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
> FraudRisk <- mutate(FraudRisk,IsHighRisk)
> Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
> Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
> Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Count
_Transactions_1_Day - Min_Count_Transactions_1_Day)
> FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
> library(ggplot2)
> qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
> qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binom
ial"))
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
>
```

As with a lm() type model, the summary() function can return the model output:

summary(LogisticRegressionModel)

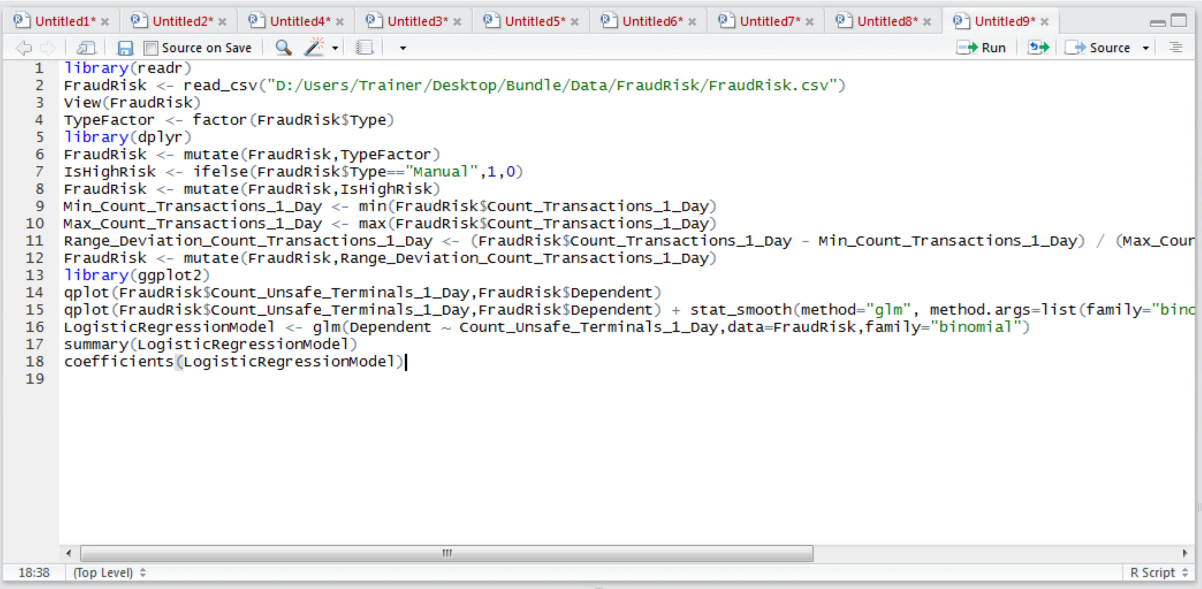
```
Untitled1* Untitled2* Untitled4* Untitled3* Untitled5* Untitled6* Untitled7* Untitled8* Untitled9*
Source on Save                                                    Run    Source
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)

17:33  (Top Level)                                                                  R Script
```

Run the line of script to console:

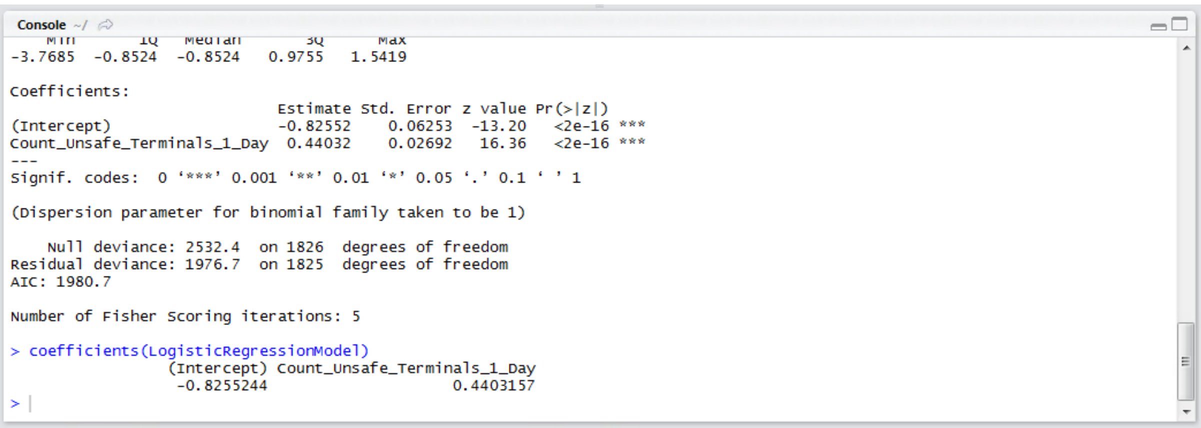
```
Console ~/
    data = FraudRisk)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.7685  -0.8524  -0.8524   0.9755   1.5419

Coefficients:
                             Estimate Std. Error z value Pr(>|z|)
(Intercept)                  -0.82552    0.06253  -13.20   <2e-16 ***
Count_Unsafe_Terminals_1_Day  0.44032    0.02692   16.36   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2532.4  on 1826  degrees of freedom
Residual deviance: 1976.7  on 1825  degrees of freedom
AIC: 1980.7

Number of Fisher Scoring iterations: 5

>
```

As with models created using the lm() function, the summary is somewhat inadequate to get the coefficients with correct precision, notwithstanding that the predict.glm() function will be used for recall:

coefficients(LogisticRegressionModel)



Run the line of script to console to output the coefficients for a manual deployment of the logistic regression model:



This procedure would naturally lead into a stepwise multiple logistic regression model, and in this example a factor as created in preceding procedures will be added with the assumption that it is the next strongest correlating factor:

```
Untitled1* Untitled2* Untitled4* Untitled3* Untitled5* Untitled6* Untitled7* Untitled8* Untitled9*
Source on Save                                                        Run   Source
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bino
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")

19:119   (Top Level)                                                                          R Script
```

Run the line of script to console:

```
Console ~/
-3.7685  -0.8524  -0.8524   0.9755   1.5419

Coefficients:
                             Estimate Std. Error z value Pr(>|z|)    
(Intercept)                  -0.82552    0.06253  -13.20   <2e-16 ***
Count_Unsafe_Terminals_1_Day  0.44032    0.02692   16.36   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2532.4  on 1826  degrees of freedom
Residual deviance: 1976.7  on 1825  degrees of freedom
AIC: 1980.7

Number of Fisher Scoring iterations: 5

> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day 
                  -0.8255244                    0.4403157 
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
> 
```

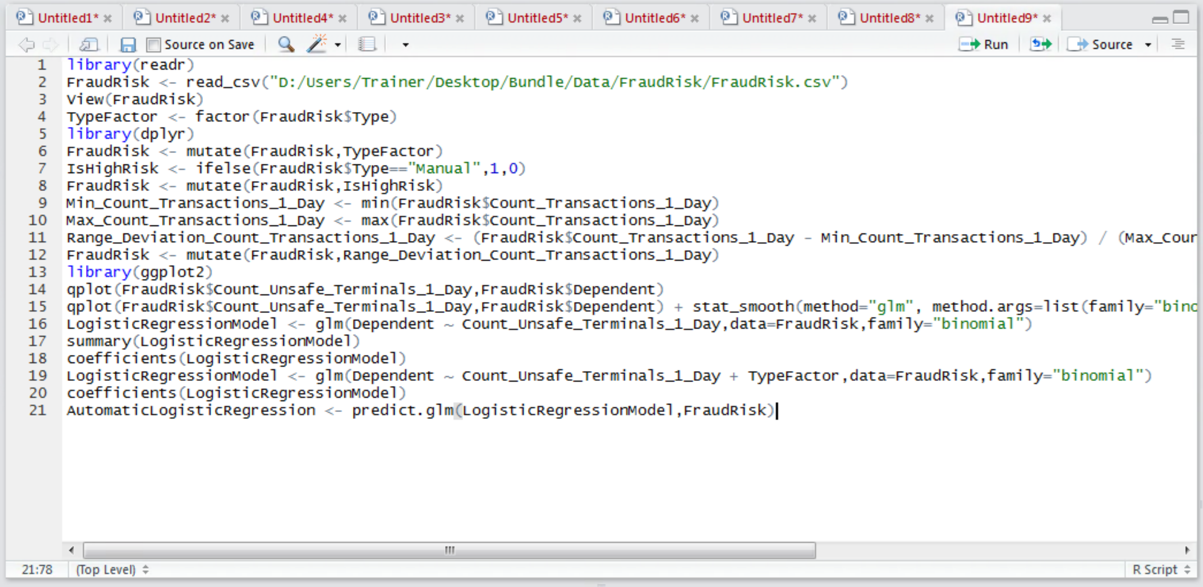
Write out the coefficients to observe the treatment of each different state inside the factor TypeFactor:

```
Console ~/
                             Estimate Std. Error z value Pr(>|z|)    
(Intercept)                  -0.82552    0.06253  -13.20   <2e-16 ***
Count_Unsafe_Terminals_1_Day  0.44032    0.02692   16.36   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2532.4  on 1826  degrees of freedom
Residual deviance: 1976.7  on 1825  degrees of freedom
AIC: 1980.7

Number of Fisher Scoring iterations: 5

> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day 
                  -0.8255244                    0.4403157 
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day             TypeFactorManual              TypeFactorSwipe 
                  -1.1538985                    0.2956514                    1.0340266                    1.8307673 
> 
```

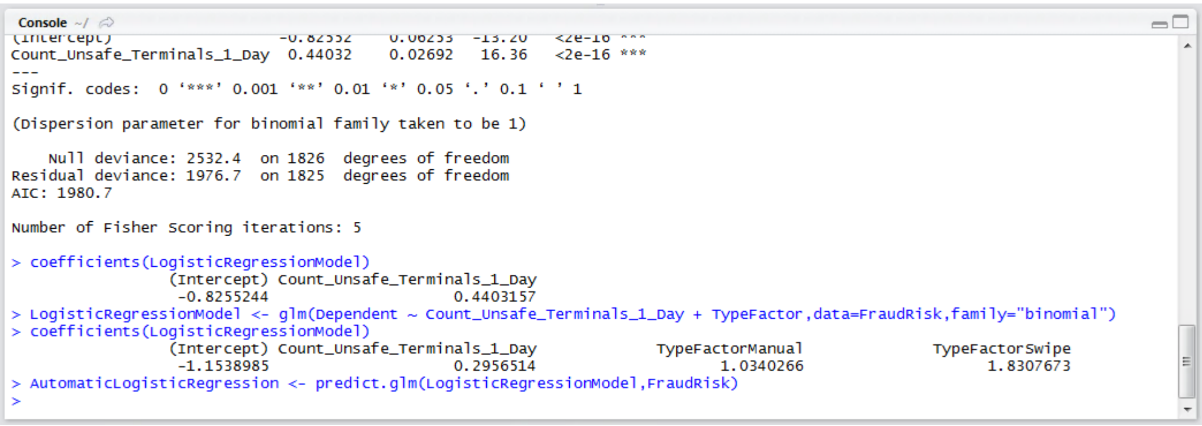
Procedure 5: Recalling a Logistic Regression Model.

It is fairly self-explanatory to deploy a logistic model, recall is performed in the same manner as a linear regression model and as described in procedure 91. As with the lm() product, the glm() model has a predict.gml() function to create a prediction for all values in a data frame. The signature bears stark resemblance to that of the predict.lm() function:

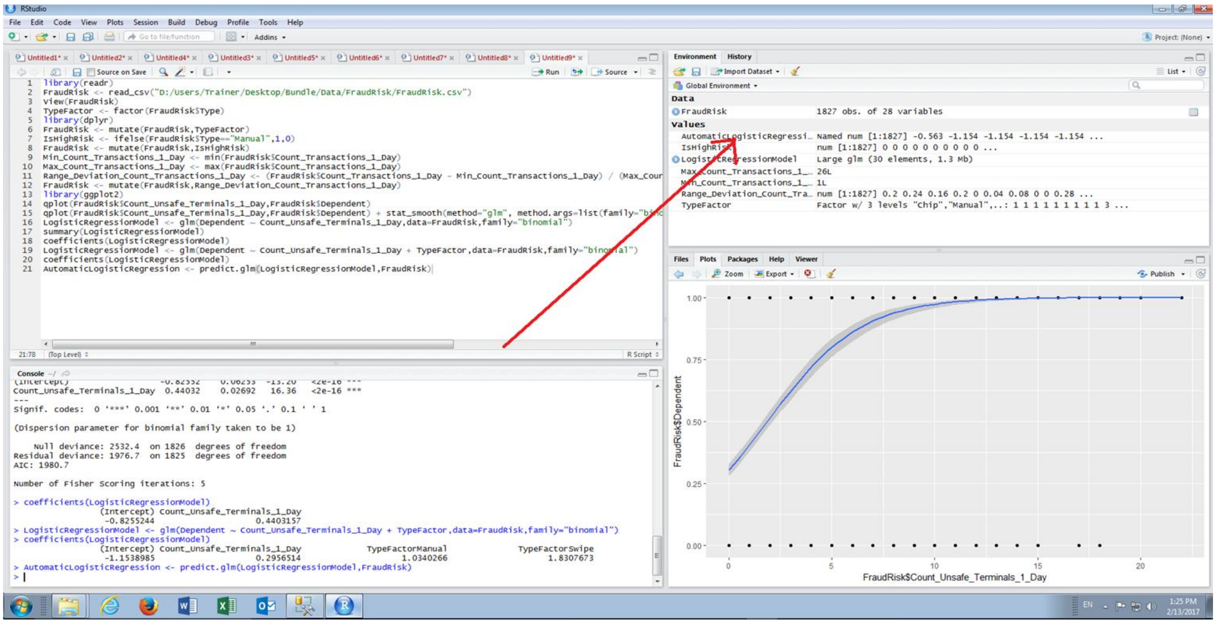
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)



Run the line of script to console:

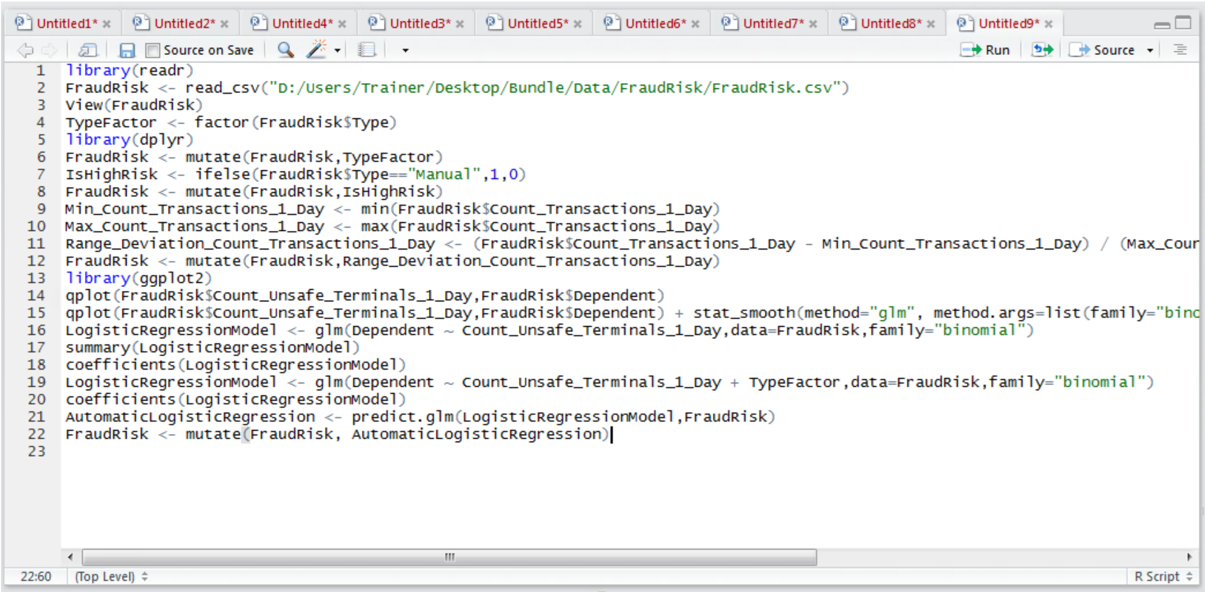


It can be seen that a new vector has been created in the environment pane which will contain the predictions for each entry in the FraudRisk Data Frame:

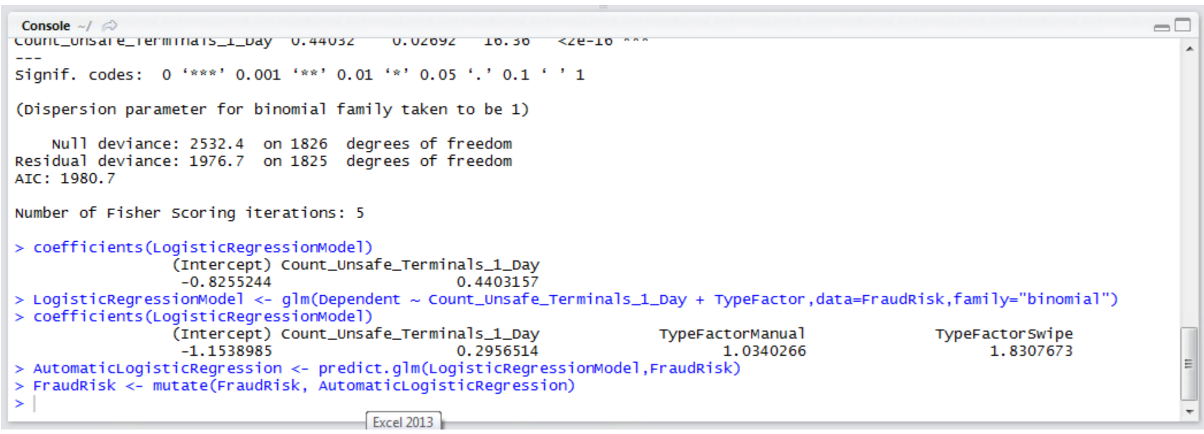


For completeness, merge the newly created vector into the FraudRisk data frame:

FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)



Run the line of script to console:



Procedure 6: Activating Logistic Regression and Creating a Confusion Matrix.

A logistic regression model outputs values between – 5 and +5, representing zero probability to 100% percent probability. Zero would represent a 50/50 probability, anything greater than zero would denote the outcome being more likely than not.

In this example, suppose that activation is to take place based upon the balance of probabilities and anything greater than 0 should be considered as being predicted, in this example, as fraud. The ifelse() function can facilitate the creation of an activation function:

ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)

```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
```

Run the line of script to console:

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2532.4  on 1826  degrees of freedom
Residual deviance: 1976.7  on 1825  degrees of freedom
AIC: 1980.7

Number of Fisher Scoring iterations: 5

> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day
                  -0.8255244                    0.4403157
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day             TypeFactorManual              TypeFactorSwipe
                  -1.1538985                    0.2956514                    1.0340266                    1.8307673
> AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
> FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
> ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
> 
```

For completeness merge the Activated Logistic Regression model into the fraud risk data frame:

FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)

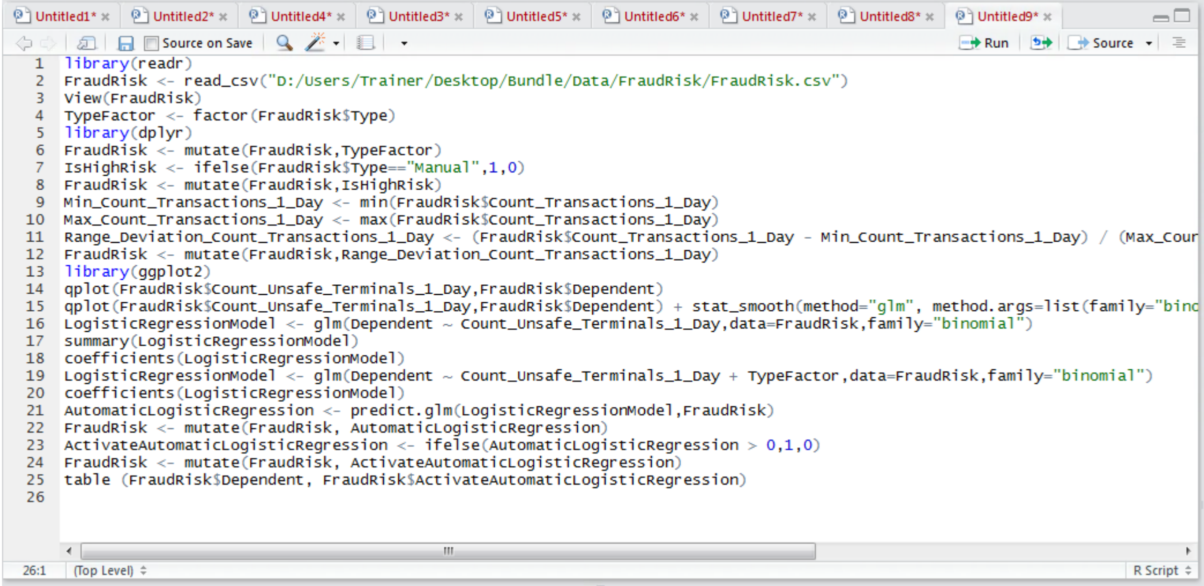
```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
```

Run the line of script to console:

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2532.4  on 1826  degrees of freedom
Residual deviance: 1976.7  on 1825  degrees of freedom
AIC: 1980.7

Number of Fisher Scoring iterations: 5

> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day
                  -0.8255244                    0.4403157
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day             TypeFactorManual              TypeFactorSwipe
                  -1.1538985                    0.2956514                    1.0340266                    1.8307673
> AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
> FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
> ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
> FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
> 
```

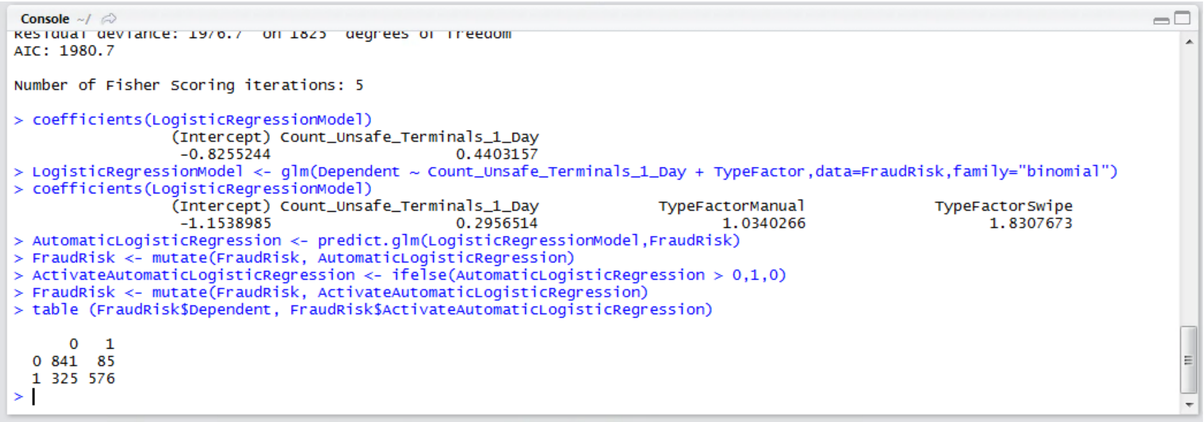
To create a confusion matrix using the table() function based upon the predicted \ ActivateAutomaticLogisticRegression vs the Actual \ Dependent variable:

table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)



```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
```

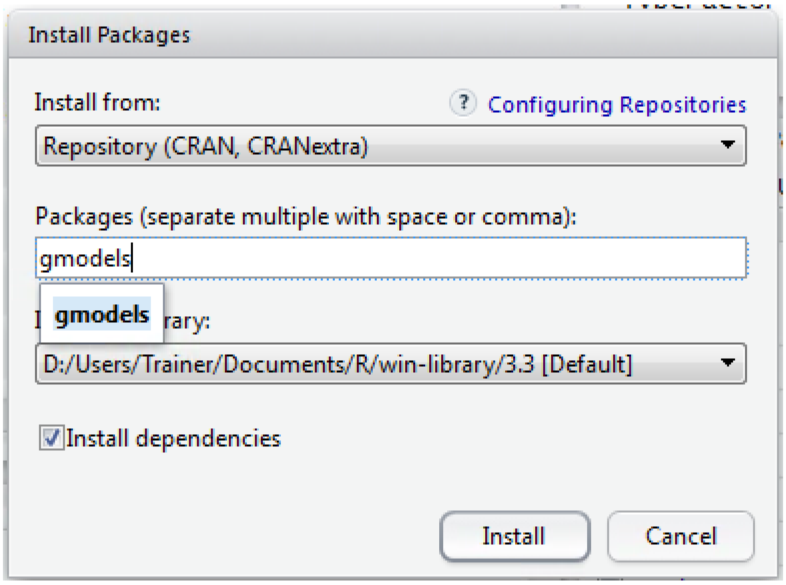
Run the line of script to console to output the confusion matrix:



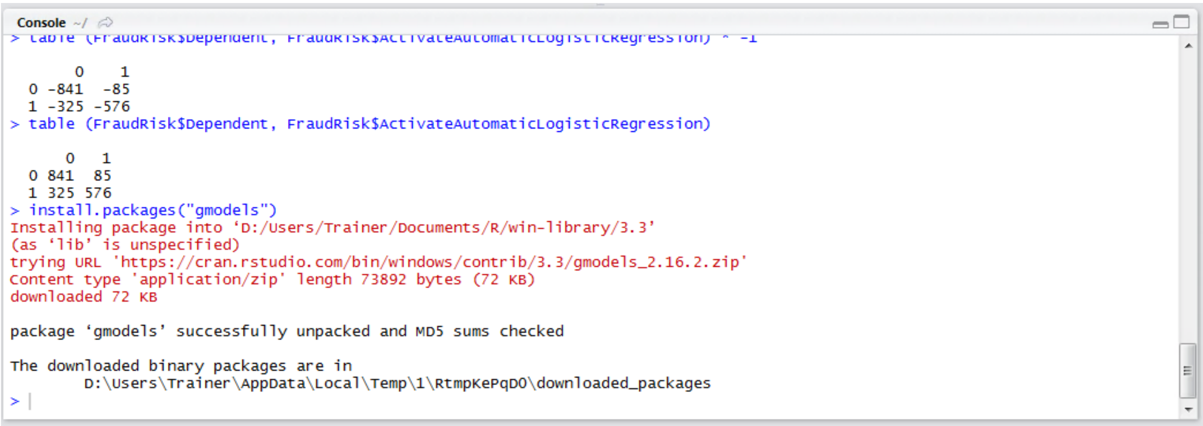
```
AIC: 1980.7

Number of Fisher Scoring iterations: 5

> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day
                  -0.8255244                    0.4403157
> LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
> coefficients(LogisticRegressionModel)
                 (Intercept) Count_Unsafe_Terminals_1_Day             TypeFactorManual              TypeFactorSwipe
                  -1.1538985                    0.2956514                    1.0340266                    1.8307673
> AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
> FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
> ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
> FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
> table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)

      0   1
  0 841  85
  1 325 576
>
```

In this example, it can be seen that of 901 records in total, 576 were judged to be fraudulent by the model and were in fraudulent in actuality, some 63.9% a figure for which improvement should be sought via stepwise logistic regression.

The process of calculating the performance of the confusion matrix in this manner is quite laborious and there exist several packages that help layout the confusion matrix with more readily available performance measures. Install the gmodels package:



Click install to both download and install:



Once the gmodels library is installed it needs to be referenced. To create the confusion matrix, the line of script resembles the table() function almost absolutely, except making use of the CrossTable() function of the gmodels package:

library("gmodels")

CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)

```r
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
```

Run the line of script to console:

```
                   | FraudRisk$ActivateAutomaticLogisticRegression
FraudRisk$Dependent |         0 |         1 | Row Total |
-------------------|-----------|-----------|-----------|
                 0 |       841 |        85 |       926 |
                   |   105.776 |   186.588 |           |
                   |     0.908 |     0.092 |     0.507 |
                   |     0.721 |     0.129 |           |
                   |     0.460 |     0.047 |           |
-------------------|-----------|-----------|-----------|
                 1 |       325 |       576 |       901 |
                   |   108.711 |   191.765 |           |
                   |     0.361 |     0.639 |     0.493 |
                   |     0.279 |     0.871 |           |
                   |     0.178 |     0.315 |           |
-------------------|-----------|-----------|-----------|
      Column Total |      1166 |       661 |      1827 |
                   |     0.638 |     0.362 |           |
-------------------|-----------|-----------|-----------|

>
```

It can be seen that a confusion matrix has been created in much the same manner except for it has created the summary statistics across both axis of the table.

Procedure 7: Output Logistic Regression Model as Probability.

The logistic regression output ranges from –5 to +5, yet oftentimes it is substantially more intuitive to present this output as a probability. The formula to convert a logistic regression output to a probability is:

P = exp(Ouput) / (1+exp(Ouput))

It follows that vector arithmetic can be used, simply swapping the output with a vector of values created by the logistic regression model:

```
library(readr)
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cour
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="binc
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
```

Run the line of script to console:

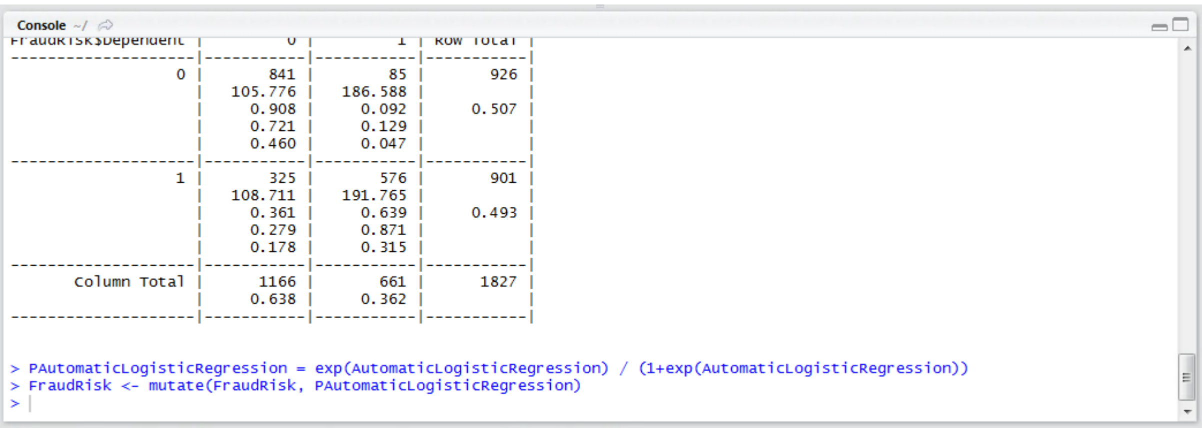
```
                    | FraudRisk$ActivateAutomaticLogisticRegression
FraudRisk$Dependent |         0 |         1 | Row Total |
--------------------|-----------|-----------|-----------|
                  0 |       841 |        85 |       926 |
                    |   105.776 |   186.588 |           |
                    |     0.908 |     0.092 |     0.507 |
                    |     0.721 |     0.129 |           |
                    |     0.460 |     0.047 |           |
--------------------|-----------|-----------|-----------|
                  1 |       325 |       576 |       901 |
                    |   108.711 |   191.765 |           |
                    |     0.361 |     0.639 |     0.493 |
                    |     0.279 |     0.871 |           |
                    |     0.178 |     0.315 |           |
--------------------|-----------|-----------|-----------|
       Column Total |      1166 |       661 |      1827 |
                    |     0.638 |     0.362 |           |
--------------------|-----------|-----------|-----------|


> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
>
```

For completeness merge the probability values into the FraudRisk data frame:

FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)

```
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
View(FraudRisk)
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
```

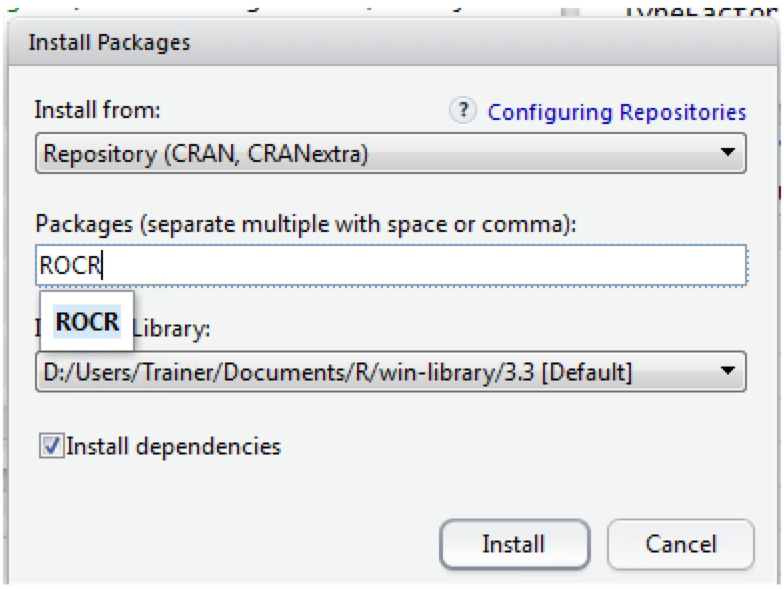
Run the line of script to console:



Procedure 8: Creating a ROC Curve.

The ROCR package provides a set of functions that simplifies the process of appraising the performance of classification models, comparing the actual outcome with a probability prediction. It can be noted that although a logistic regression model outputs between -5 and + 5, procedure 101 converted this value to an intuitive probability.

Firstly, install the ROCR package from the RStudio package installation utility.



Click install to proceed with the installation:

```
Console ~/
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/caTools_1.17.1.zip'
Content type 'application/zip' length 284328 bytes (277 KB)
downloaded 277 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/gplots_3.0.1.zip'
Content type 'application/zip' length 511932 bytes (499 KB)
downloaded 499 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/ROCR_1.0-7.zip'
Content type 'application/zip' length 152168 bytes (148 KB)
downloaded 148 KB

package 'bitops' successfully unpacked and MD5 sums checked
package 'gtools' successfully unpacked and MD5 sums checked
package 'gdata' successfully unpacked and MD5 sums checked
package 'caTools' successfully unpacked and MD5 sums checked
package 'gplots' successfully unpacked and MD5 sums checked
package 'ROCR' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\RtmpAXi3GX\downloaded_packages
> |
```

Reference the ROC Library:

library(ROCR)

```
Untitled1* × Untitled2* × Untitled4* × Untitled3* × Untitled5* × Untitled6* × Untitled7* × Untitled8* × Untitled9* ×
Source on Save                                                    Run   Source
TypeFactor <- factor(FraudRisk$Type)
library(dplyr)
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
library(ROCR)

30:14  (Top Level) ÷                                                                                            R Script ÷
```

Run the block of script to console:

```
Console ~/
                    |    108.711 |    191.763 |            |
                    |      0.361 |      0.639 |      0.493 |
                    |      0.279 |      0.871 |            |
                    |      0.178 |      0.315 |            |
--------------------|------------|------------|------------|
       Column Total |       1166 |        661 |       1827 |
                    |      0.638 |      0.362 |            |
--------------------|------------|------------|------------|

> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
> FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
> library(ROCR)
Loading required package: gplots

Attaching package: 'gplots'

The following object is masked from 'package:stats':

    lowess

> |
```

Two vectors and inputs are needed to create a visualisation, the first is the predictions expressed as a probability, the second being the actual outcome. In this example, it will be the vector FraudRisk$ PAutomaticLogisticRegression And FraudRisk$Dependent. To create the predictions object in ROCR:

ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)

```
Console ~/
                   |     0.361 |     0.639 |     0.493 |
                   |     0.279 |     0.871 |           |
                   |     0.178 |     0.315 |           |
-------------------|-----------|-----------|-----------|
      Column Total |      1166 |       661 |      1827 |
                   |     0.638 |     0.362 |           |
-------------------|-----------|-----------|-----------|

> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
> FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
> library(ROCR)
Loading required package: gplots

Attaching package: 'gplots'

The following object is masked from 'package:stats':

    lowess

> ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
> |
```

Once the prediction object has been created it needs to be morphed into a performance object using the performance() function. The performance function takes the prediction object yet also an indication as to the performance measures to be used, in this case true positive rate (tpr) vs false positive rate (fpr). The performance function outputs an object that can be used in conjunction with the base graphic plot() function:

ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")

```
Untitled1* × Untitled2* × Untitled4* × Untitled3* × Untitled5* × Untitled6* × Untitled7* × Untitled8* × Untitled9* ×
Source on Save                                                              Run    Source
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
library(ROCR)
ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")|

32:82   (Top Level) ÷                                                                                     R Script ÷
```

Run the line of script to console:

```
                   |     0.279 |     0.871 |           |
                   |     0.178 |     0.315 |           |
-------------------|-----------|-----------|-----------|
      Column Total |      1166 |       661 |      1827 |
                   |     0.638 |     0.362 |           |
-------------------|-----------|-----------|-----------|


> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
> FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
> library(ROCR)
Loading required package: gplots

Attaching package: ‘gplots’

The following object is masked from ‘package:stats’:

    lowess

> ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
> ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
> 
```

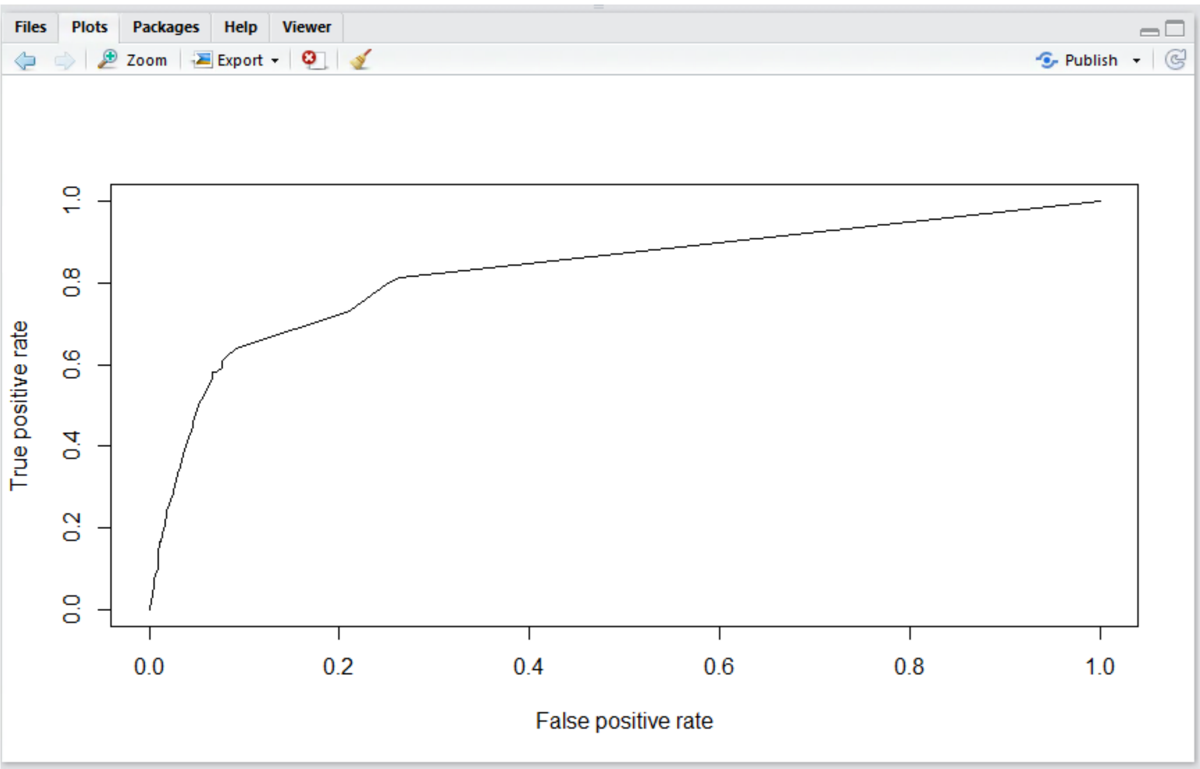
Simply plot the ROCRPerformance object by passing as an argument to the plot() base graphic function:

```
FraudRisk <- mutate(FraudRisk,TypeFactor)
IsHighRisk <- ifelse(FraudRisk$Type=="Manual",1,0)
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
library(ROCR)
ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
plot(ROCRPerformance)
```

Run the line of script to console:

```
                   |     0.178 |     0.315 |           |
-------------------|-----------|-----------|-----------|
      Column Total |      1166 |       661 |      1827 |
                   |     0.638 |     0.362 |           |
-------------------|-----------|-----------|-----------|


> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
> FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
> library(ROCR)
Loading required package: gplots

Attaching package: ‘gplots’

The following object is masked from ‘package:stats’:

    lowess

> ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
> ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
> plot(ROCRPerformance)
> 
```

It can be seen that a curve plot has been created in the plots window in RStudio:

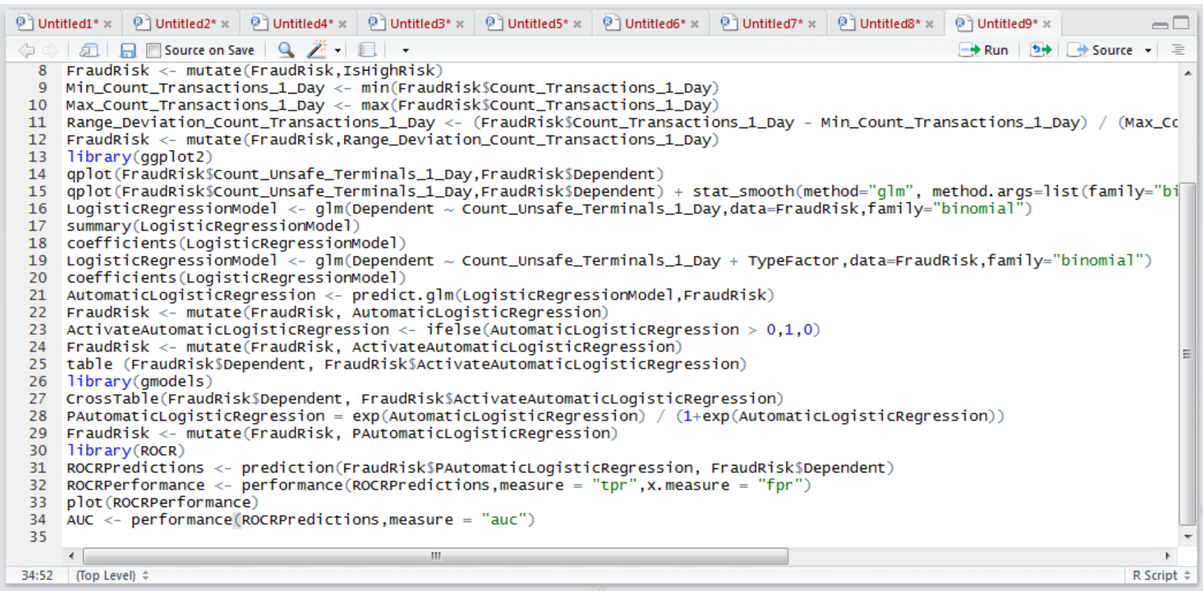


It can be seen that the line is not diagonal, leading to an inference that the model has some predictive power.

Procedure 9: Grading the ROC Performance with AUC.

Visually the plot created in procedure 102 suggests a that the model created has some predictive power. A more succinct method to measure model performance is the Area Under Curve statistics which can be calculated with ease by requesting "auc" as the measure to the performance object:

AUC <- performance(ROCRPredictions,measure = "auc")



```
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
library(ROCR)
ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
plot(ROCRPerformance)
AUC <- performance(ROCRPredictions,measure = "auc")
```

Run the line of script to console:

```
--------------------|-----------|-----------|-----------|
       Column Total |      1166 |       661 |      1827 |
                    |     0.638 |     0.362 |           |
--------------------|-----------|-----------|-----------|


> PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
> FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
> library(ROCR)
Loading required package: gplots

Attaching package: ‘gplots’

The following object is masked from ‘package:stats’:

    lowess

> ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
> ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
> plot(ROCRPerformance)
> AUC <- performance(ROCRPredictions,measure = "auc")
> 
```

To write out the contents of the AUC object:

AUC

```
FraudRisk <- mutate(FraudRisk,IsHighRisk)
Min_Count_Transactions_1_Day <- min(FraudRisk$Count_Transactions_1_Day)
Max_Count_Transactions_1_Day <- max(FraudRisk$Count_Transactions_1_Day)
Range_Deviation_Count_Transactions_1_Day <- (FraudRisk$Count_Transactions_1_Day - Min_Count_Transactions_1_Day) / (Max_Cc
FraudRisk <- mutate(FraudRisk,Range_Deviation_Count_Transactions_1_Day)
library(ggplot2)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent)
qplot(FraudRisk$Count_Unsafe_Terminals_1_Day,FraudRisk$Dependent) + stat_smooth(method="glm", method.args=list(family="bi
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day,data=FraudRisk,family="binomial")
summary(LogisticRegressionModel)
coefficients(LogisticRegressionModel)
LogisticRegressionModel <- glm(Dependent ~ Count_Unsafe_Terminals_1_Day + TypeFactor,data=FraudRisk,family="binomial")
coefficients(LogisticRegressionModel)
AutomaticLogisticRegression <- predict.glm(LogisticRegressionModel,FraudRisk)
FraudRisk <- mutate(FraudRisk, AutomaticLogisticRegression)
ActivateAutomaticLogisticRegression <- ifelse(AutomaticLogisticRegression > 0,1,0)
FraudRisk <- mutate(FraudRisk, ActivateAutomaticLogisticRegression)
table (FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
library(gmodels)
CrossTable(FraudRisk$Dependent, FraudRisk$ActivateAutomaticLogisticRegression)
PAutomaticLogisticRegression = exp(AutomaticLogisticRegression) / (1+exp(AutomaticLogisticRegression))
FraudRisk <- mutate(FraudRisk, PAutomaticLogisticRegression)
library(ROCR)
ROCRPredictions <- prediction(FraudRisk$PAutomaticLogisticRegression, FraudRisk$Dependent)
ROCRPerformance <- performance(ROCRPredictions,measure = "tpr",x.measure = "fpr")
plot(ROCRPerformance)
AUC <- performance(ROCRPredictions,measure = "auc")
AUC
```

Run the line of script to console:

```
An object of class "performance"
Slot "x.name":
[1] "None"

Slot "y.name":
[1] "Area under the ROC curve"

Slot "alpha.name":
[1] "none"

Slot "x.values":
list()

Slot "y.values":
[[1]]
[1] 0.827767


Slot "alpha.values":
list()

> 
```

The value to gravitate towards is the y.values, which will have a value ranging between 0.5 and 1:

```
Console ~/
An object of class "performance"
Slot "x.name":
[1] "None"

Slot "y.name":
[1] "Area under the ROC curve"

Slot "alpha.name":
[1] "none"

Slot "x.values":
list()

Slot "y.values":
[[1]]
[1] 0.827767

Slot "alpha.values":
list()

>
```

In this example, the AUC value is 0.827767 which suggests that the model has an excellent utility. By way of grading, AUC scores would correspond:

- A: Outstanding > 0.9
- B: Excellent > 0.8 and <= 0.9
- C: Acceptable > 0.7 and <= 0.8
- D: Poor > 0.6 and <= 0.7
- E: Junk > 0.5 and <= 0.6

Module 10: Splits, Probability and Decision Trees.

Probability and product is a fairly radical departure from regression based techniques and form the foundation creating decision trees. However, as a convenient stepping stone to splitting, there is a hybrid technique which uses the concept of splitting based on the standard deviation. This module intends to introduce the concept of splitting data into homogenous groups, as best can be, with a view to creating decision trees on this data.

This module uses two different datasets. For the purposes of creating regression trees Bundle\Data\Equity\Abstracted\FDX\PC_FDX_Close_200x1D_Close_50x1D_10.csv which contains data that has already been abstracted for the FedEx stock on the NYSE.

For the purposes of creating C5 decision trees the dataset Bundle\Data\CreditRisk\CreditRisk.csv is used.

Start with a new script and import both datasets as per procedure 46:

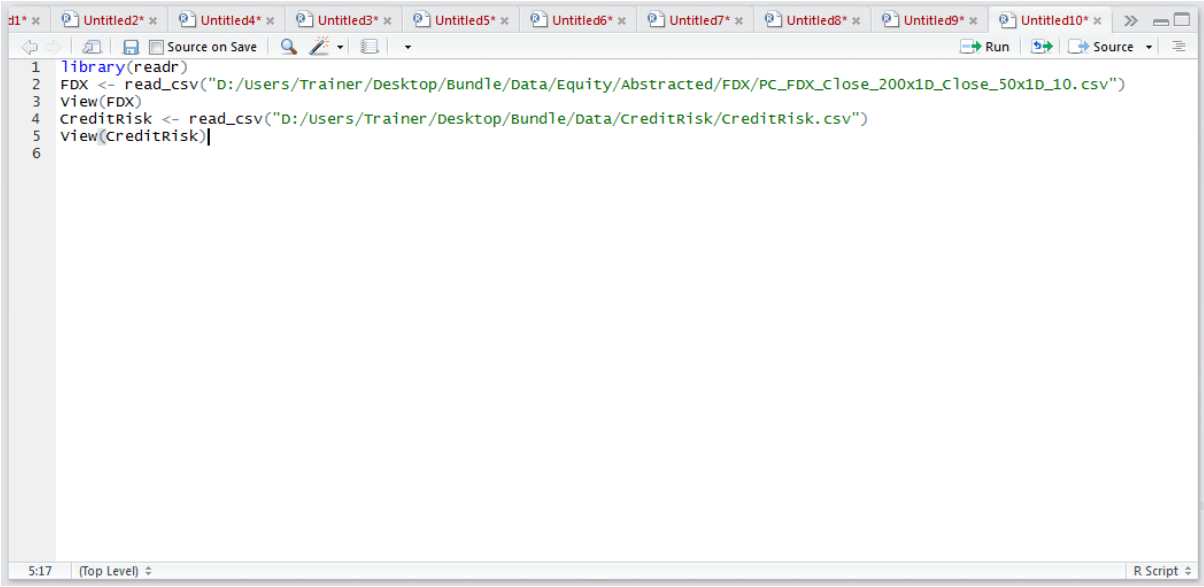
library(readr)

FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")

View(FDX)

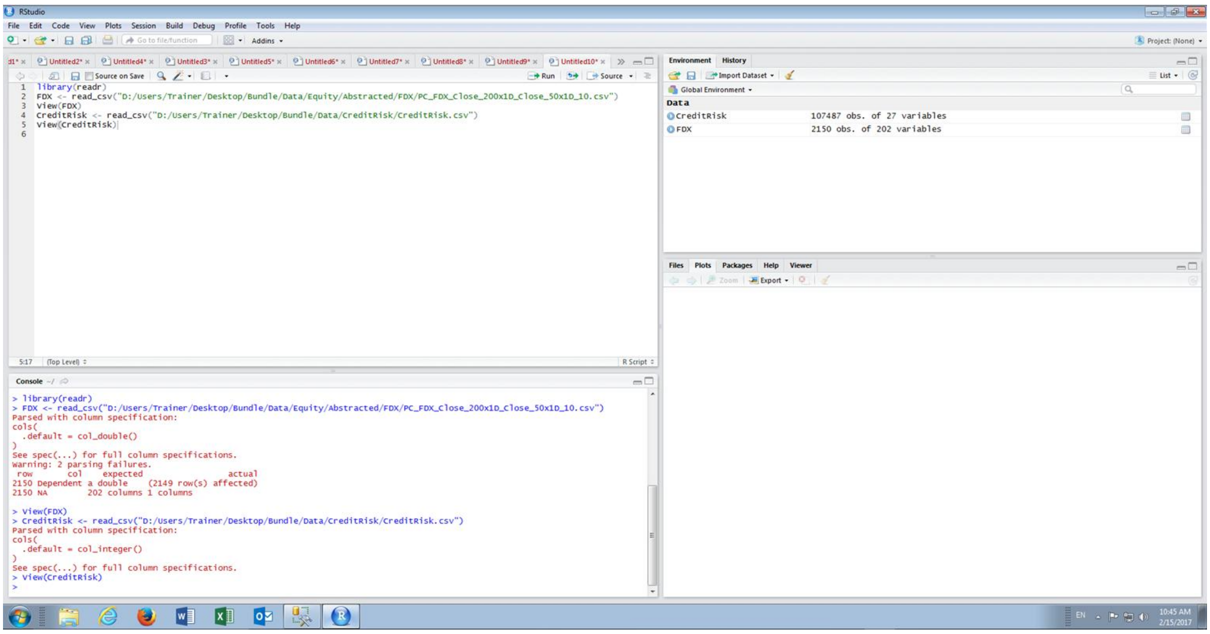
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")

View(CreditRisk)



Run the block of script to console:

It can be seen that there are now two data frames available in the environment pane for use in the subsequent procedures:

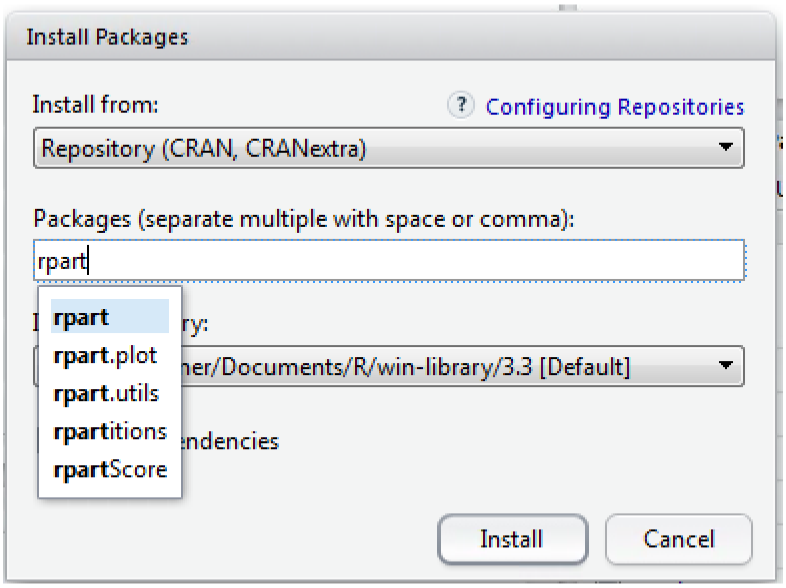


Note however that in loading the CreditRisk data frame, the read.csv() function of base R has been used and NOT the read_csv() function of the library readr. The functions of readr are many times faster and more efficient than that of the base R functions but will never convert character strings to factors, instead presenting them as character vectors.

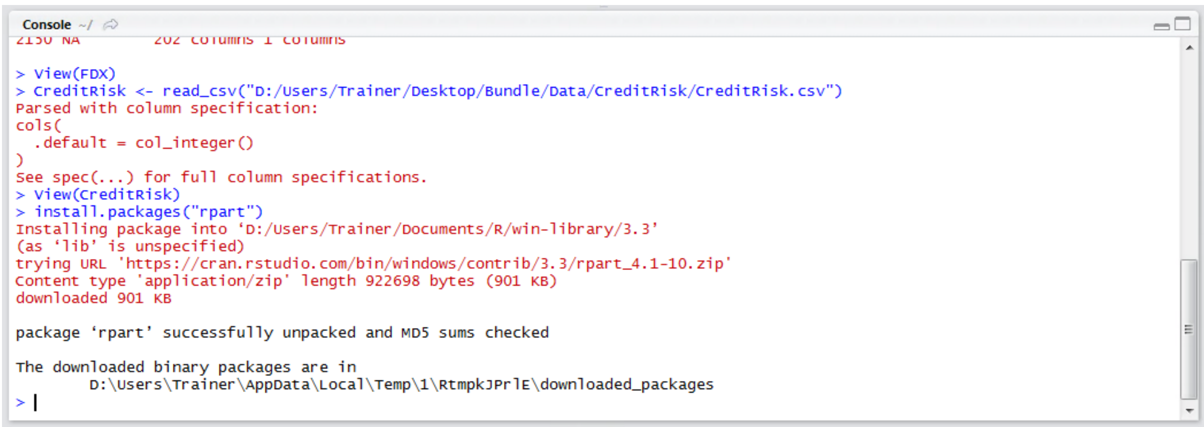
As the CreditRisk data frame is going to be used exclusively for classification, it is very useful that any character strings are inferred as factors and will save a large amount of time in pre-processing. It follows as a rule of thumb, use the read_csv() almost universally, unless the intention is to convert character strings to factors in which case use read.csv() function.

Procedure 1: Create a Decision Tree using rpart.

Firstly it is nessecary to install the rpart package:

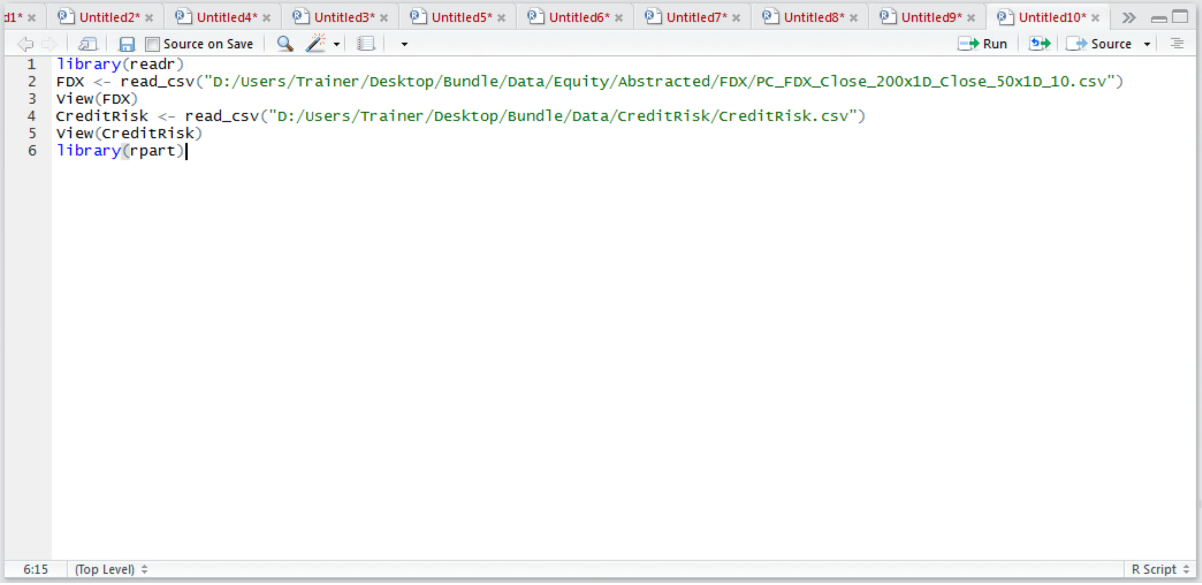


Click the install button to download and install the package:

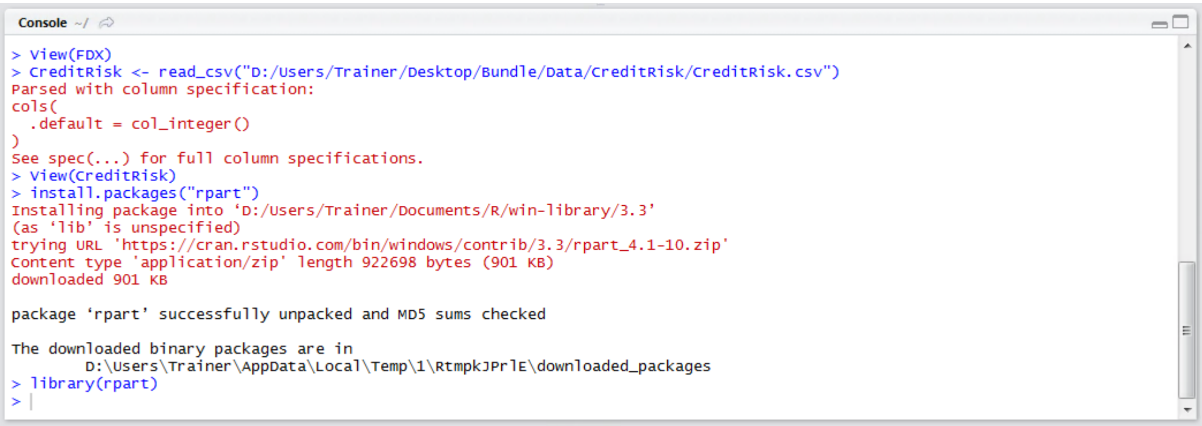


Load the lirary rpart:

library(rpart)

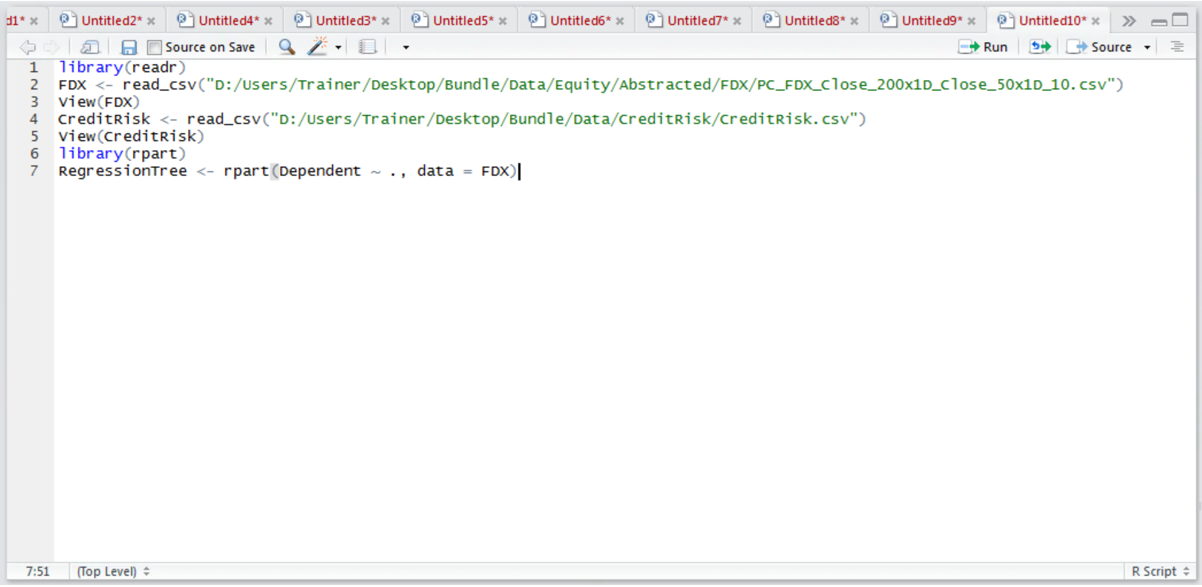


Run the line of script to console:

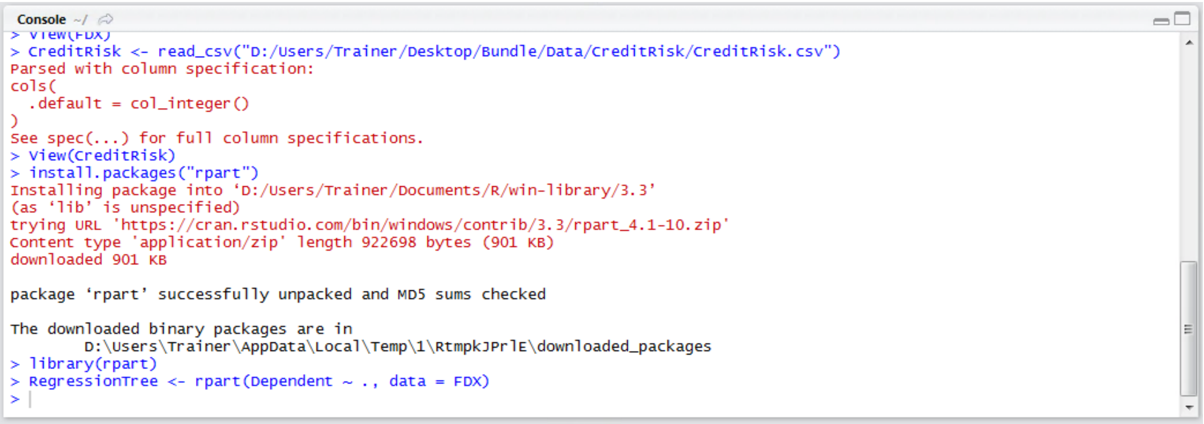


Unlike in regression procedures, where it is nessecary to be selective about the independent variables being passed to the model, regression trees do not nessecarily require any variable selection as they are much better at producting models on very large feature sets, as such can be instructed to use all indepenent variables. To train a regression tree, use the rpart() function, specify the dependent variable and the independent variables:

RegressionTree <- rpart(Dependent ~ ., data = FDX)

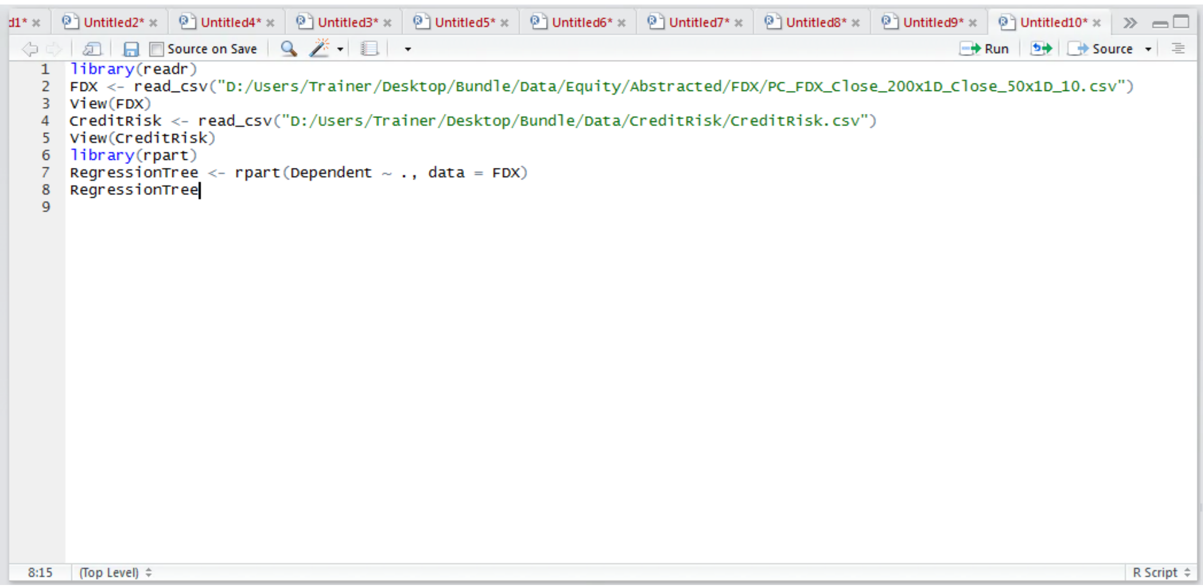


Run the line of script to console:



A regression tree has been created and saved to the RegressionTree object. To retrieve information about the splits and tree simply:

RegressionTree



Run the line of script to console:

```
Console ~/
      40) PearsonCorrelation_4< -0.7000403 163  1.06929800 -0.071549070
        80) Median_3_ZScore< 0.3452189 114  0.36904200 -0.110894600 *
        81) Median_3_ZScore>=0.3452189 49  0.11318810  0.019989610 *
      41) PearsonCorrelation_4>=-0.7000403 1052  4.93724300  0.016500630
        82) Range_3_PearsonCorrelation>=0.3518519 142  0.44377260 -0.060156460 *
        83) Range_3_PearsonCorrelation< 0.3518519 910  3.52882600  0.028462500
         166) PointStep_9_ZScore>=0.2657032 323  0.77074260 -0.009661027 *
         167) PointStep_9_ZScore< 0.2657032 587  2.03031700  0.049440180
           334) Skew_3< -0.3590697 154  0.62272810  0.004195887 *
           335) Skew_3>=-0.3590697 433  0.98022450  0.065531690 *
    21) Close_2_PearsonCorrelation< -0.7651638 41  0.09212358  0.183720000 *
  11) Range_4< 1.559354 414  3.93178400  0.081336470
    22) PointStep_1>=0.607338 66  0.22909910 -0.051621670 *
    23) PointStep_1< 0.607338 348  2.31466700  0.106552700
      46) Range_3_PearsonCorrelation< 0.108431 291  1.58303100  0.088708090
        92) Min_3_PearsonCorrelation< -0.5127741 25  0.19446850 -0.055712650 *
        93) Min_3_PearsonCorrelation>=-0.5127741 266  0.81812180  0.102281500 *
      47) Range_3_PearsonCorrelation>=0.108431 57  0.16590520  0.197654000 *
3) Range_1< 1.095992 56  1.08038500  0.257308000
  6) Range_4_ZScore>=-1.12821 35  0.17794380  0.171248700 *
  7) Range_4_ZScore< -1.12821 21  0.21119510  0.400740200 *
>
```

A regression tree has been written out that can be interpreted as a series of if, then, else statements. In this example, the follow logic would predict a percentage price change, although there are many variations:

If Range_1>=1.095992 and Min_2>=0.1196234 and PointStep_12_PearsonCorrelation>=-0.3153543 then Forecast is -0.149363700

```
Console ~/
     denotes terminal node

 1) root 2149 26.50178000  0.016412940
   2) Range_1>=1.095992 2093 22.08474000  0.009967582
     4) TypicalValue_4_PearsonCorrelation< -0.6301845 423  5.41561900 -0.061559880
       8) Min_2< 0.1196234 149  2.37821100 -0.144294800
        16) PointStep_16_ZScore>=-1.199479 92  0.95450130 -0.209382900
          32) Min_3_PearsonCorrelation>=-0.7110189 81  0.40032710 -0.237454300 *
          33) Min_3_PearsonCorrelation< -0.7110189 11  0.02033586 -0.002674954 *
        17) PointStep_16_ZScore< -1.199479 57  0.40487950 -0.039240450 *
       9) Min_2>=0.1196234 274  1.46286700 -0.016568970
        18) PointStep_12_PearsonCorrelation>=-0.3153543 39  0.09689294 -0.149363700 *
        19) PointStep_12_PearsonCorrelation< -0.3153543 235  0.56409440  0.005469300 *
     5) TypicalValue_4_PearsonCorrelation>=-0.6301845 1670 13.95682000  0.028085020
      10) Range_4>=1.559354 1256  8.46407900  0.010532390
        20) Close_2_PearsonCorrelation>=-0.7651638 1215  7.10070600  0.004688198
          40) PearsonCorrelation_4< -0.7000403 163  1.06929800 -0.071549070
            80) Median_3_ZScore< 0.3452189 114  0.36904200 -0.110894600 *
            81) Median_3_ZScore>=0.3452189 49  0.11318810  0.019989610 *
          41) PearsonCorrelation_4>=-0.7000403 1052  4.93724300  0.016500630
            82) Range_3_PearsonCorrelation>=0.3518519 142  0.44377260 -0.060156460 *
            83) Range_3_PearsonCorrelation< 0.3518519 910  3.52882600  0.028462500
             166) PointStep_9_ZScore>=0.2657032 323  0.77074260 -0.009661027 *
```

The rtree() function has though suggested a wide range of potential rules, the endpoints being denoted by a *. It is though important to understand the performance of each one these endpoints to propose implementation of these rules. To establish the error rates of these endpoints, use the summary() function passing the RegressionTree object:

summary(RegressionTree)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
```

Run the line of script to console:

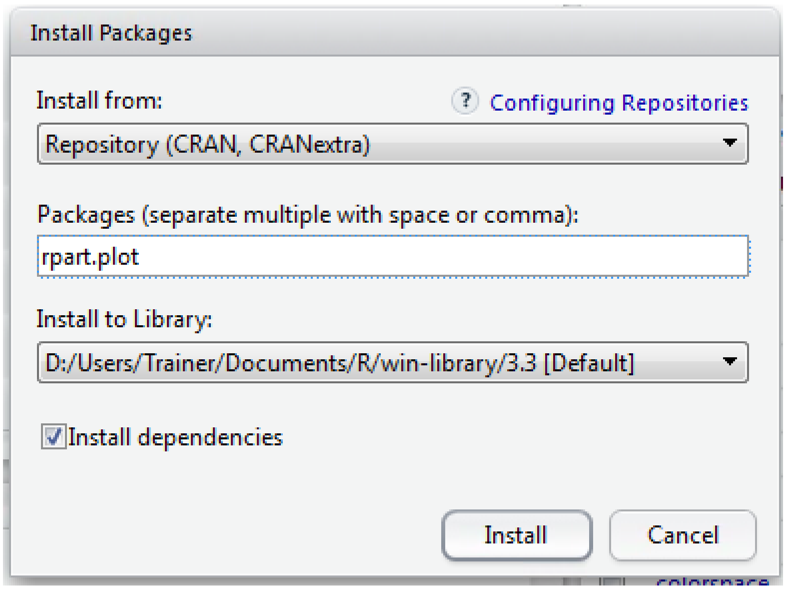
```
  mean=0.04944018, MSE=0.003458803
  left son=334 (154 obs) right son=335 (433 obs)
  Primary splits:
      Skew_3                      < -0.3590697  to the left,  improve=0.2104916, (0 missing)
      Max_1_ZScore                < 15.28321    to the right, improve=0.1668905, (0 missing)
      Skew_3_PearsonCorrelation   < -0.4286609  to the left,  improve=0.1626707, (0 missing)
      Skew_2                      < -0.09811816 to the left,  improve=0.1588004, (0 missing)
      Median_2_ZScore             < -2.492116   to the left,  improve=0.1467861, (0 missing)
  Surrogate splits:
      Close_1_ZScore  < -2.427568  to the left,  agree=0.860, adj=0.468, (0 split)
      Skew_2          < -0.4842811 to the left,  agree=0.855, adj=0.448, (0 split)
      Median_2_ZScore < -2.544749  to the left,  agree=0.850, adj=0.429, (0 split)
      Range_1_ZScore  < -0.3156007 to the left,  agree=0.848, adj=0.422, (0 split)
      Skew_4          < -0.6440575 to the left,  agree=0.847, adj=0.416, (0 split)

Node number 334: 154 observations
  mean=0.004195887, MSE=0.004043689

Node number 335: 433 observations
  mean=0.06553169, MSE=0.002263798

> |
```

Notice in the regression tree output, each node is labelled and in this example, node 18 was referenced. By searching for this node in the summary output, the error rate can be determined:

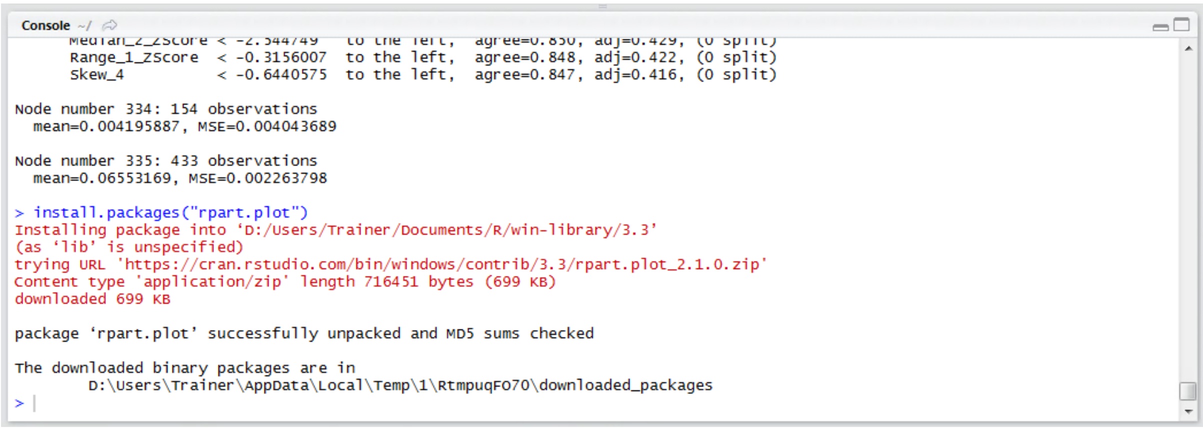
```
  mean=-0.2093829, MSE=0.01037501
  left son=32 (81 obs) right son=33 (11 obs)
  Primary splits:
      Min_3_PearsonCorrelation         < -0.7110189  to the right, improve=0.5592851, (0 missing)
      PointStep_18_PearsonCorrelation  < -0.4502016  to the right, improve=0.5592851, (0 missing)
      Max_1_PearsonCorrelation         < -0.595132   to the left,  improve=0.5592851, (0 missing)
      PointStep_4_PearsonCorrelation   < -0.6199754  to the left,  improve=0.5369947, (0 missing)
      Min_4                            < 0.446191    to the left,  improve=0.4948073, (0 missing)
  Surrogate splits:
      Max_1_PearsonCorrelation          < -0.595132   to the left,  agree=1.000, adj=1.000, (0 split)
      PointStep_18_PearsonCorrelation   < -0.4502016  to the right, agree=1.000, adj=1.000, (0 split)
      Median_1_PearsonCorrelation       < -0.5308417  to the left,  agree=0.989, adj=0.909, (0 split)
      TrimmedMean_1_PearsonCorrelation  < -0.5492711  to the left,  agree=0.989, adj=0.909, (0 split)
      Min_1_PearsonCorrelation          < -0.5481415  to the left,  agree=0.989, adj=0.909, (0 split)

Node number 17: 57 observations
  mean=-0.03924045, MSE=0.007103149

Node number 18: 39 observations
  mean=-0.1493637, MSE=0.002484434

Node number 19: 235 observations
```

Procedure 2: Visualise a rpart Decision Tree.

Once familiar with the output of a regression tree, it becomes an informative means to create business rules. Quite often however, for the purposes of communication, it is more satisfying to create a visualisation. A package called rpart.plot is available for the purposes of translating regression trees to a visualisation. Start by installing the rpart.ploy package:

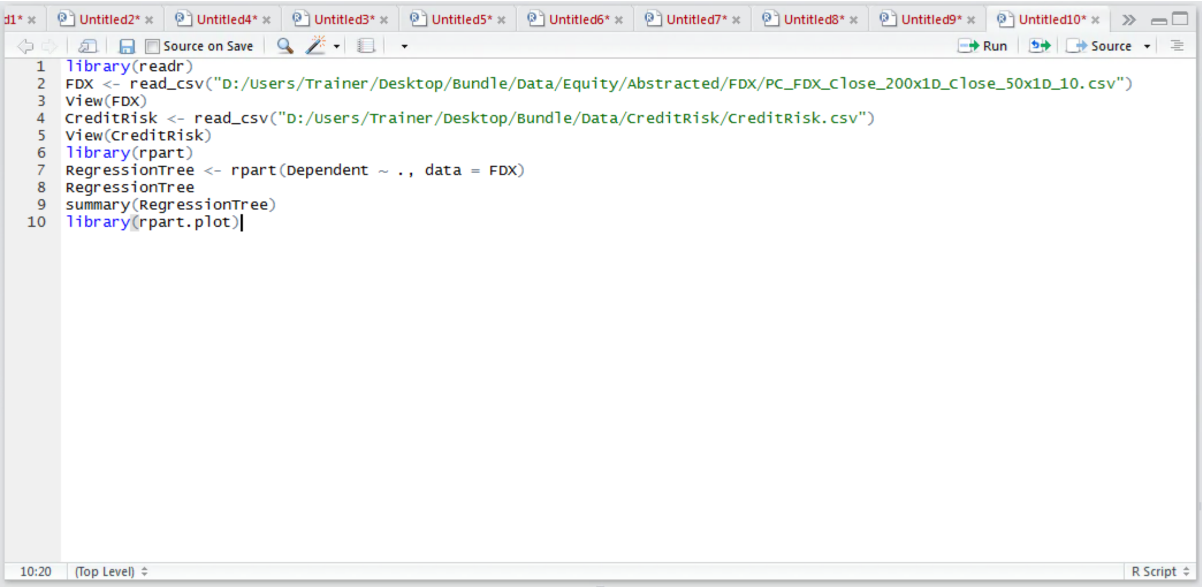


Click install to download and install the package:

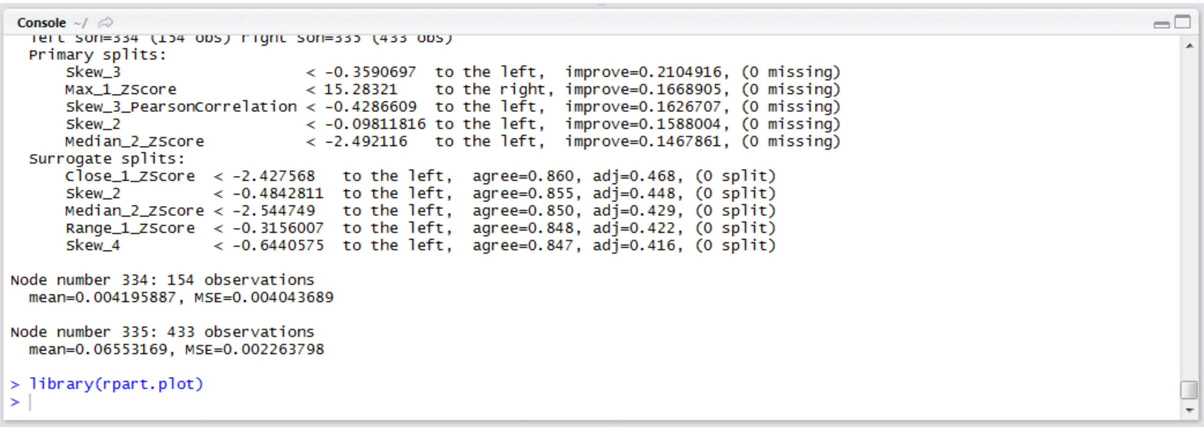


Reference the library:

library(rpart.plot)

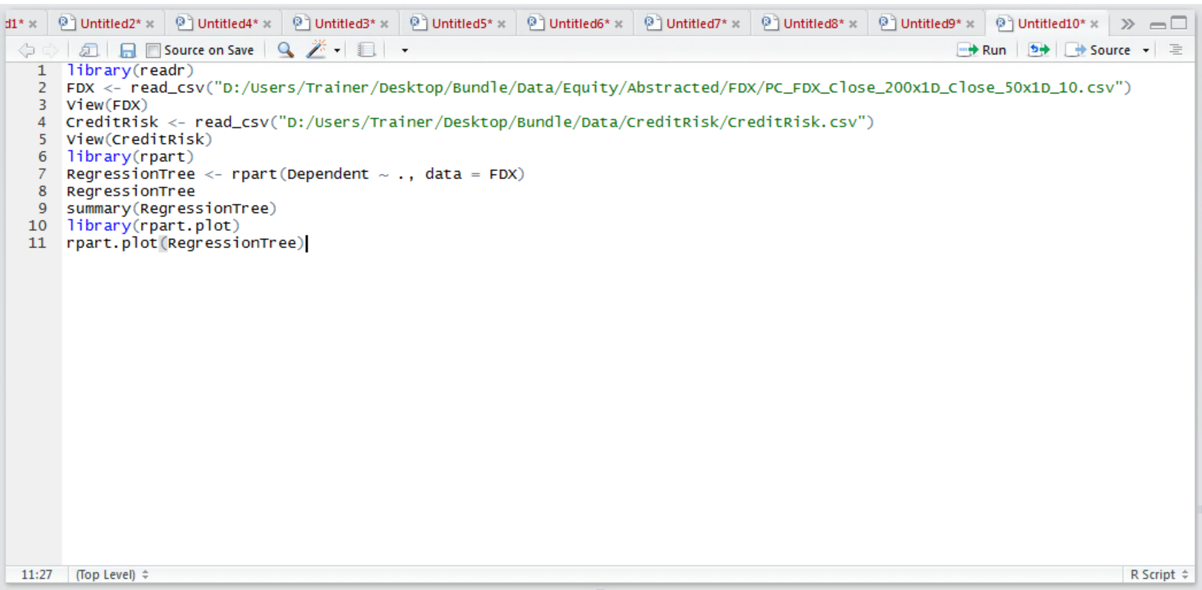


Run the line of script to console:

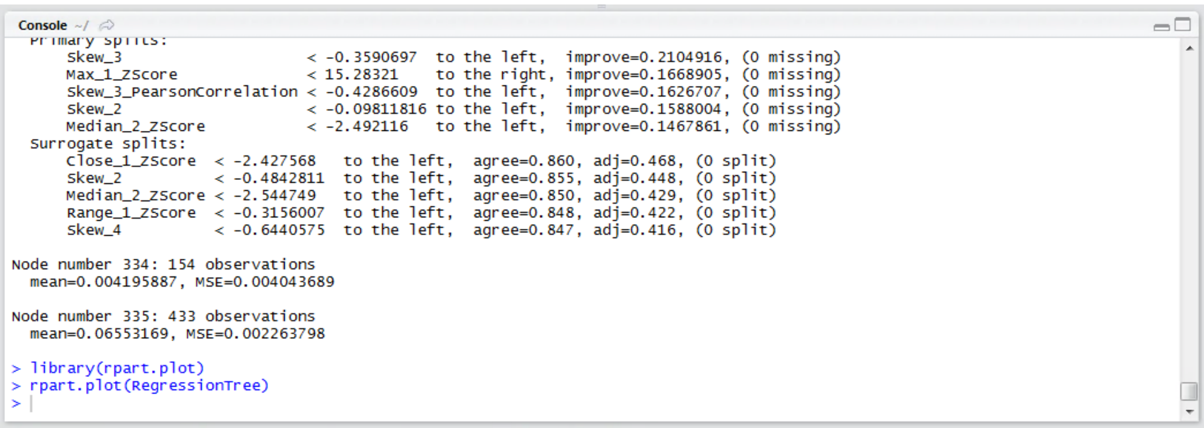


To transpose the Regression Tree to a plot, simply pass it as an argument to the rpart.plot() function:

rpart.plot(RegressionTree)

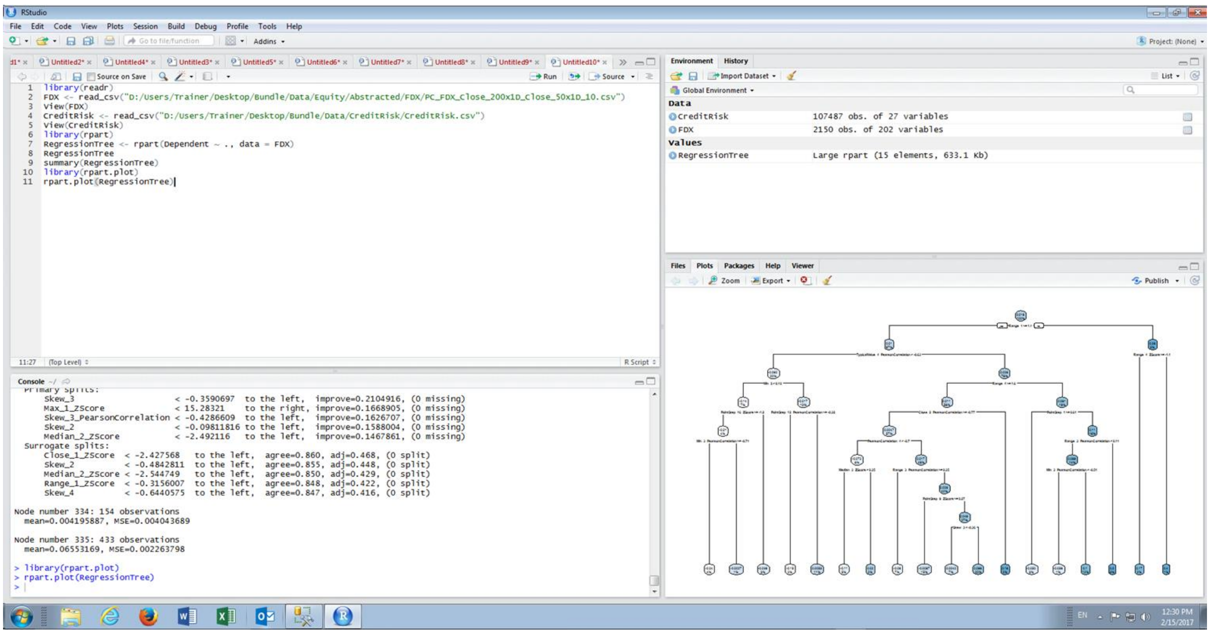


Run the line of script to console:

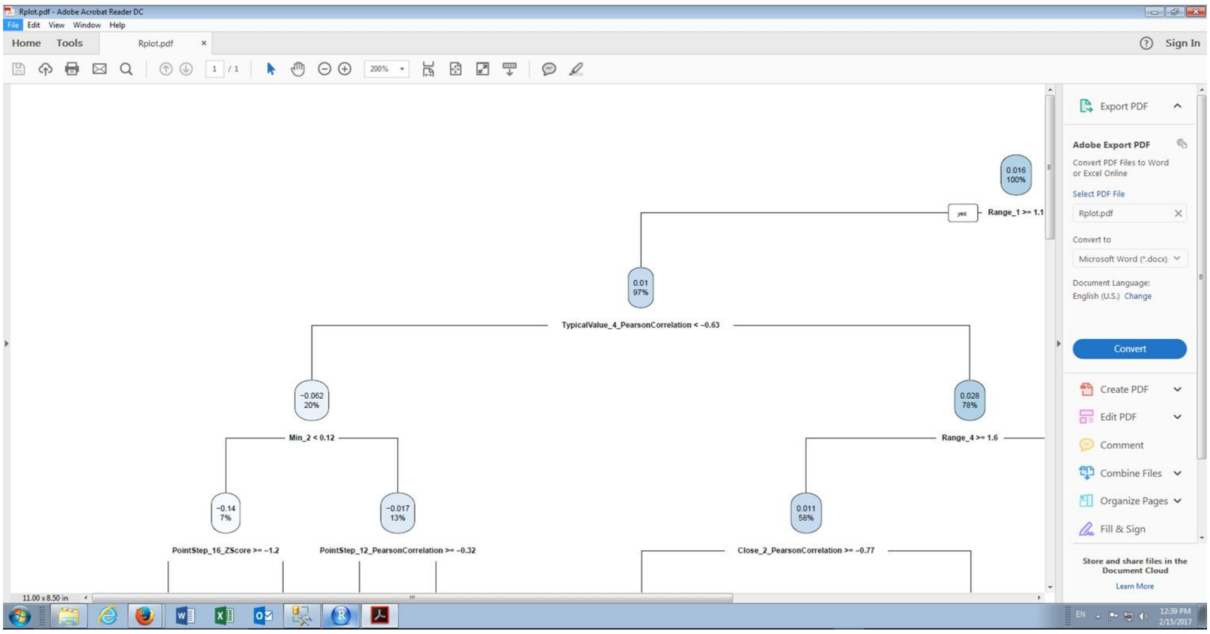


```
  Primary splits:
      Skew_3                     < -0.3590697  to the left,  improve=0.2104916, (0 missing)
      Max_1_ZScore               < 15.28321    to the right, improve=0.1668905, (0 missing)
      Skew_3_PearsonCorrelation  < -0.4286609  to the left,  improve=0.1626707, (0 missing)
      Skew_2                     < -0.09811816 to the left,  improve=0.1588004, (0 missing)
      Median_2_ZScore            < -2.492116   to the left,  improve=0.1467861, (0 missing)
  Surrogate splits:
      Close_1_ZScore  < -2.427568   to the left,  agree=0.860, adj=0.468, (0 split)
      Skew_2          < -0.4842811  to the left,  agree=0.855, adj=0.448, (0 split)
      Median_2_ZScore < -2.544749   to the left,  agree=0.850, adj=0.429, (0 split)
      Range_1_ZScore  < -0.3156007  to the left,  agree=0.848, adj=0.422, (0 split)
      Skew_4          < -0.6440575  to the left,  agree=0.847, adj=0.416, (0 split)

Node number 334: 154 observations
  mean=0.004195887, MSE=0.004043689

Node number 335: 433 observations
  mean=0.06553169, MSE=0.002263798

> library(rpart.plot)
> rpart.plot(RegressionTree)
>
```

It can be seen that a complex visualisation has been created in the plots window of R Studio:



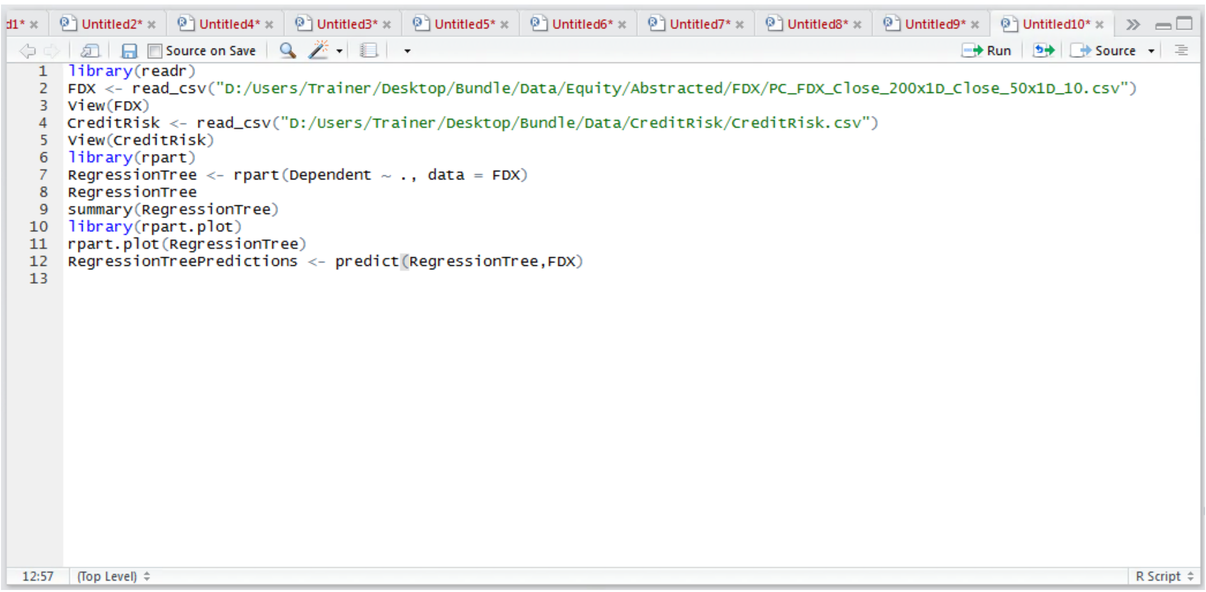
The visualisation is exceptionally hard to interpret for a large regression tree; hence it will likely need to be exported to a PDF or Image file to use a zoom function:



Procedure 3: Recalling a rpart() Decision Tree.

As with regression and most of the predictive analytics tools presented in this document, the predict() function can take the RegressionTree object in conjunction with a data frame, then return the predictions. To create predictions using the RegressionTree model and the FDX dataset:

RegressionTreePredictions <- predict(RegressionTree,FDX)



Run the line of script to console:

```
Console ~/
      Skew_3                     < -0.3390697  to the left,  improve=0.2104916, (0 missing)
      Max_1_ZScore               < 15.28321    to the right, improve=0.1668905, (0 missing)
      Skew_3_PearsonCorrelation < -0.4286609  to the left,  improve=0.1626707, (0 missing)
      Skew_2                     < -0.09811816 to the left,  improve=0.1588004, (0 missing)
      Median_2_ZScore            < -2.492116   to the left,  improve=0.1467861, (0 missing)
  Surrogate splits:
      Close_1_ZScore  < -2.427568   to the left,  agree=0.860, adj=0.468, (0 split)
      Skew_2          < -0.4842811  to the left,  agree=0.855, adj=0.448, (0 split)
      Median_2_ZScore < -2.544749   to the left,  agree=0.850, adj=0.429, (0 split)
      Range_1_ZScore  < -0.3156007  to the left,  agree=0.848, adj=0.422, (0 split)
      Skew_4          < -0.6440575  to the left,  agree=0.847, adj=0.416, (0 split)

Node number 334: 154 observations
  mean=0.004195887, MSE=0.004043689

Node number 335: 433 observations
  mean=0.06553169, MSE=0.002263798

> library(rpart.plot)
> rpart.plot(RegressionTree)
> RegressionTreePredictions <- predict(RegressionTree,FDX)
>
```

Merge the newly created vector into the FDX data frame for completeness:

library(dplyr)

FDX <- mutate(FDX, RegressionTreePredictions)

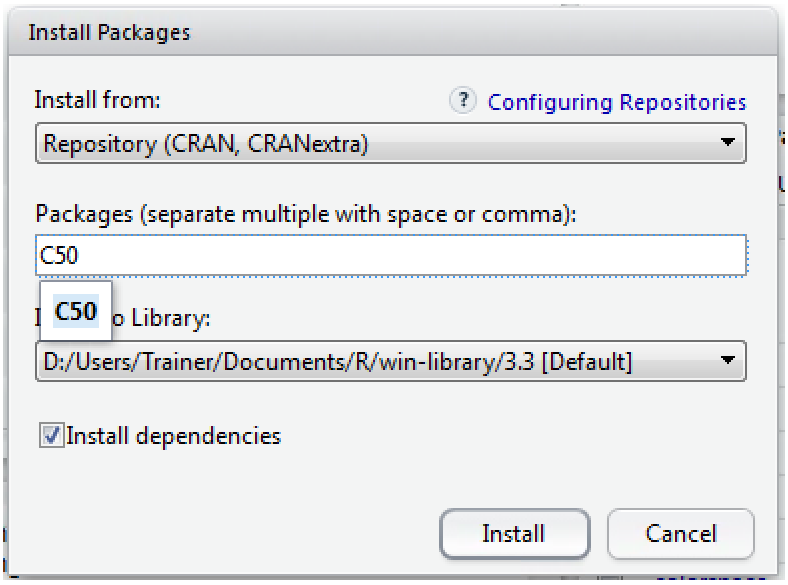
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
```

Run the block of script to console:

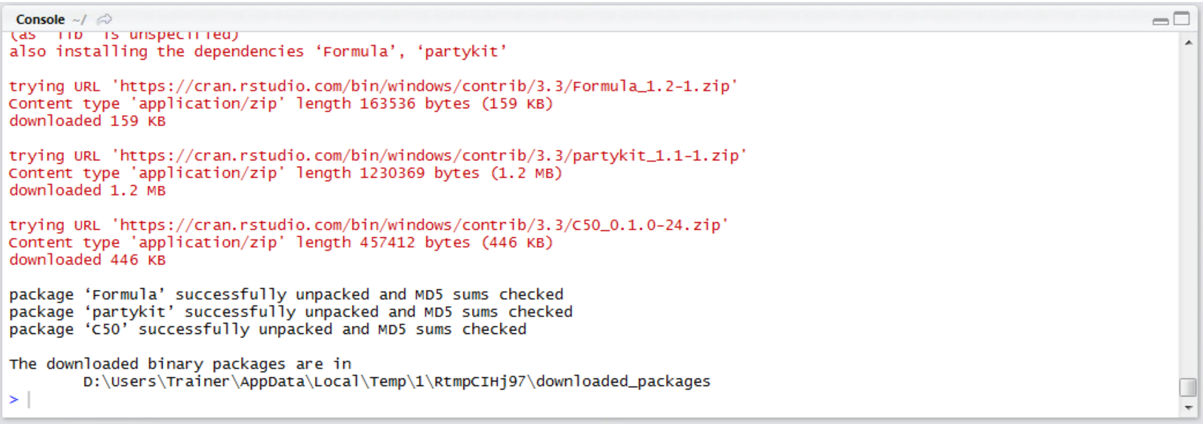
```
Console ~/
  mean=0.004195887, MSE=0.004043689

Node number 335: 433 observations
  mean=0.06553169, MSE=0.002263798

> library(rpart.plot)
> rpart.plot(RegressionTree)
> RegressionTreePredictions <- predict(RegressionTree,FDX)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FDX <- mutate(FDX, RegressionTreePredictions)
>
```

Procedure 4: Creating a C5 Decision Tree object.

Install the C50 package using RStudio:

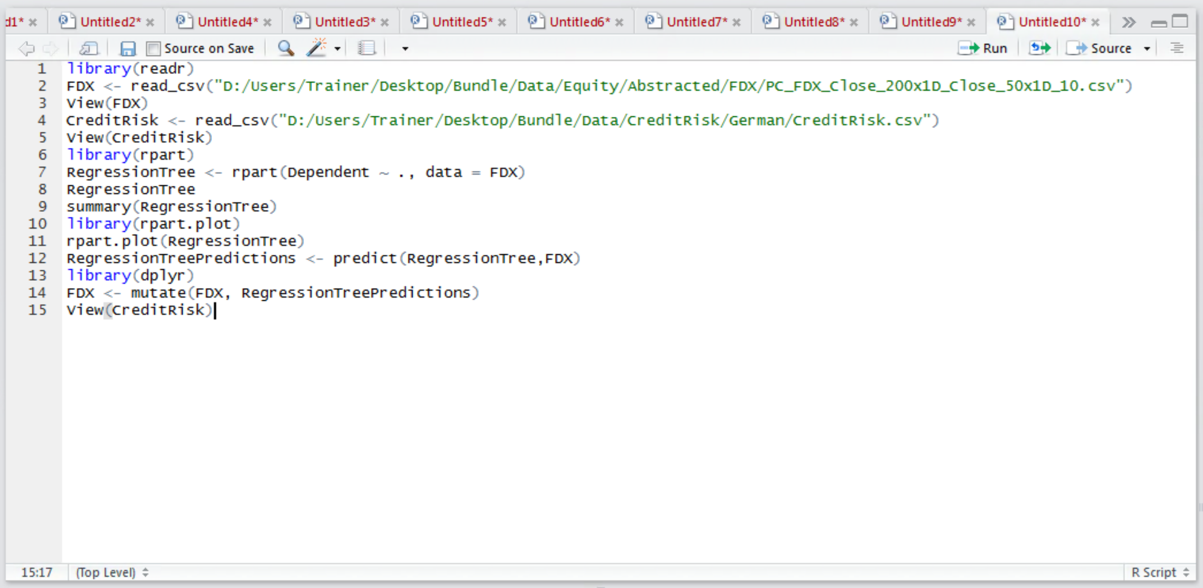


Click Install to download and install the package:



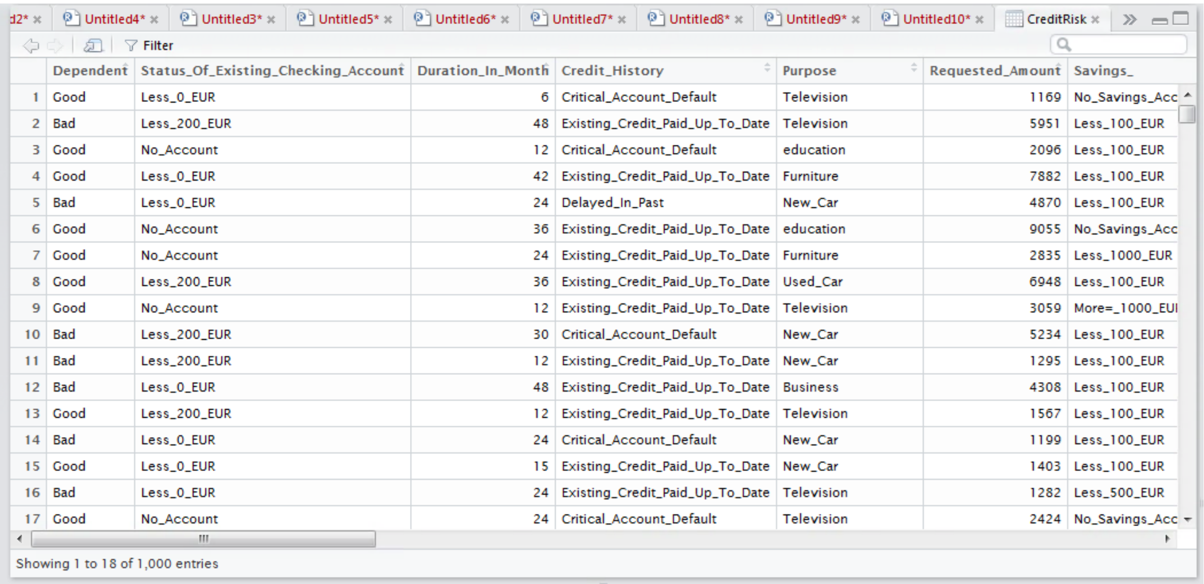
The CreditRisk data frame contains loan application data and a dependent variable which details the overall loan performance, titled Dependent for consistency. The first and most obvious difference between this data frame and those used previously is the extent to which data is categorical and string based:

View(CreditRisk)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
```

Run the line of script to console:

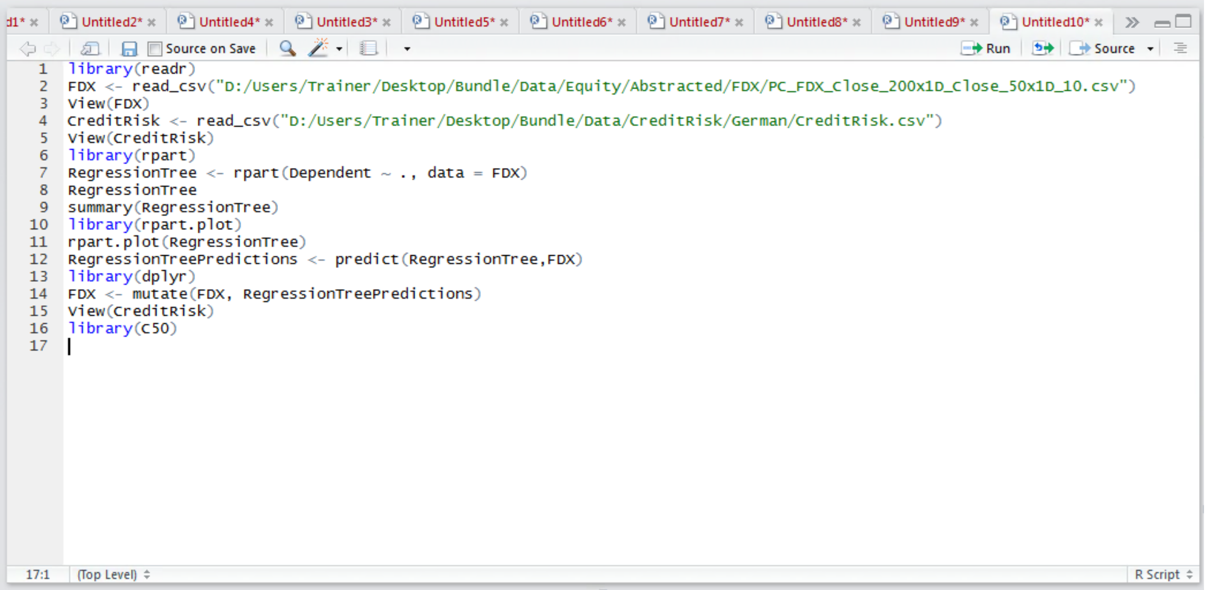


	Dependent	Status_Of_Existing_Checking_Account	Duration_In_Month	Credit_History	Purpose	Requested_Amount	Savings_
1	Good	Less_0_EUR	6	Critical_Account_Default	Television	1169	No_Savings_Acc
2	Bad	Less_200_EUR	48	Existing_Credit_Paid_Up_To_Date	Television	5951	Less_100_EUR
3	Good	No_Account	12	Critical_Account_Default	education	2096	Less_100_EUR
4	Good	Less_0_EUR	42	Existing_Credit_Paid_Up_To_Date	Furniture	7882	Less_100_EUR
5	Bad	Less_0_EUR	24	Delayed_In_Past	New_Car	4870	Less_100_EUR
6	Good	No_Account	36	Existing_Credit_Paid_Up_To_Date	education	9055	No_Savings_Acc
7	Good	No_Account	24	Existing_Credit_Paid_Up_To_Date	Furniture	2835	Less_1000_EUR
8	Good	Less_200_EUR	36	Existing_Credit_Paid_Up_To_Date	Used_Car	6948	Less_100_EUR
9	Good	No_Account	12	Existing_Credit_Paid_Up_To_Date	Television	3059	More=_1000_EUI
10	Bad	Less_200_EUR	30	Critical_Account_Default	New_Car	5234	Less_100_EUR
11	Bad	Less_200_EUR	12	Existing_Credit_Paid_Up_To_Date	New_Car	1295	Less_100_EUR
12	Bad	Less_0_EUR	48	Existing_Credit_Paid_Up_To_Date	Business	4308	Less_100_EUR
13	Good	Less_200_EUR	12	Existing_Credit_Paid_Up_To_Date	Television	1567	Less_100_EUR
14	Bad	Less_0_EUR	24	Critical_Account_Default	New_Car	1199	Less_100_EUR
15	Good	Less_0_EUR	15	Existing_Credit_Paid_Up_To_Date	New_Car	1403	Less_100_EUR
16	Bad	Less_0_EUR	24	Existing_Credit_Paid_Up_To_Date	Television	1282	Less_500_EUR
17	Good	No_Account	24	Critical_Account_Default	Television	2424	No_Savings_Acc

Showing 1 to 18 of 1,000 entries

Emphasising, the dataset is far more categorical in nature. To begin training a C5 Decision Tree load the library:

library(C50)



Run the line of script to console:

The input parameters to the C5.0() function, which is used to train a decision tree, is slightly different to that observed in preceding procedures. A data frame containing only the independent variables (no dependent variable), then a vector containing the dependent variable is required to train a model and in this regard, it differs from many of the other procedures in this guide.

In this example, the CreditRisk data frame contains both dependent and independent variables and needs splitting, in this case using negative subsetting to negate the first column then referencing the dependent variable explicitly:

C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
```

Run the line of script to console:

```
> library(rpart.plot)
> rpart.plot(RegressionTree)
> RegressionTreePredictions <- predict(RegressionTree,FDX)
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FDX <- mutate(FDX, RegressionTreePredictions)
> View(CreditRisk)
> library(C50)
> C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
>
```

The C5 decision tree has now been created and stored in the C50Tree object. To view basic information about the tree:

C50Tree

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
```

Run the line of script to console:

```
Console ~/
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> FDX <- mutate(FDX, RegressionTreePredictions)
> View(CreditRisk)
> library(C50)
> C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
> C50Tree

Call:
C5.0.default(x = CreditRisk[-1], y = CreditRisk$Dependent)

Classification Tree
Number of samples: 1000
Number of predictors: 20

Tree size: 72

Non-standard options: attempt to group attributes

> 
```

Use the summary() function to output the C5 decision tree and view the logic required to implement the classification tool:

summary(C50Tree)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
```

Run the line of script to console:

```
Console ~/
        54.30% Credit_History
        48.40% Duration_In_Month
        35.70% Savings_
        27.80% Guarantors
        24.10% Purpose
        15.70% Other_Credit
        14.00% Present_Employment_Since
         6.80% Housing
         5.50% Present_Residence_Since
         5.00% Job
         3.70% Status_Sex
         2.90% Telephone
         2.40% Installment_Percentage_Of_Disposable_Income
         2.30% Age
         1.50% Security
         1.10% Number_Of_Existing_Credits_At_This_Bank
         0.60% Dependent_Persons


Time: 0.0 secs

> 
```

The summary output is overwhelming, however, scrolling up through the pane of results reveals the decision tree:

```
Class specified by attribute `outcome'

Read 1000 cases (21 attributes) from undefined.data

Decision tree:

Status_Of_Existing_Checking_Account in {More=_200_EUR,
:                                       No_Account}: Good (457/60)
Status_Of_Existing_Checking_Account in {Less_0_EUR,Less_200_EUR}:
:...Credit_History in {All_Paid,No_Credit_Open_Or_All_Paid}:
    :...Housing in {Free,Security}: Bad (32/4)
    :   Housing = Owner:
    :   :...Purpose in {Domestic_Appliances,Retraining,
    :       :           Used_Car}: Good (4)
    :       Purpose in {education,Repairs,Television,Used_Car0}: Bad (6/1)
    :       Purpose = New_Car:
    :       :...Duration_In_Month <= 22: Bad (7)
    :       :   Duration_In_Month > 22: Good (2)
    :       Purpose = Business:
    :       :...Telephone = No: Good (3)
    :       :   Telephone = Yes_Own_Name:
    :       :   :...Other_Credit = Bank: Good (2)
```

The interpretation of this decision tree is very similar that of a regression tree. One such branch in this example would suggest that the following scenario would yield a bad account:

If Housing = Owner AND Purpose = "New Car" AND the Loan_Duration <= 22 Months Then BAD

In the above example, out of 1000 cases, it can be seen that 7 cases had this disposition:

```
Class specified by attribute `outcome'

Read 1000 cases (21 attributes) from undefined.data

Decision tree:

Status_Of_Existing_Checking_Account in {More=_200_EUR,
:                                       No_Account}: Good (457/60)
Status_Of_Existing_Checking_Account in {Less_0_EUR,Less_200_EUR}:
:...Credit_History in {All_Paid,No_Credit_Open_Or_All_Paid}:
    :...Housing in {Free,Security}: Bad (32/4)
    :   Housing = Owner:
    :   :...Purpose in {Domestic_Appliances,Retraining,
    :       :           Used_Car}: Good (4)
    :       Purpose in {education,Repairs,Television,Used_Car0}: Bad (6/1)
    :       Purpose = New_Car:
    :       :...Duration_In_Month <= 22: Bad (7)
    :       :   Duration_In_Month > 22: Good (2)
    :       Purpose = Business:
    :       :...Telephone = No: Good (3)
    :       :   Telephone = Yes_Own_Name:
    :       :   :...Other_Credit = Bank: Good (2)
```

Scrolling down further, below the tree output, is the performance measures of the model overall:

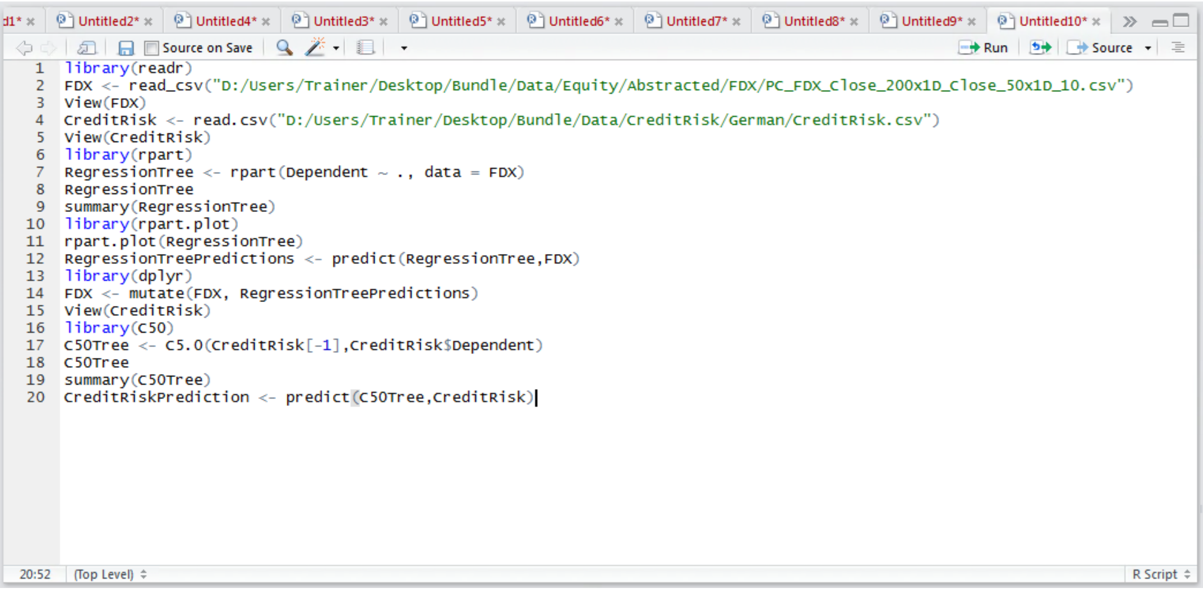
```
Evaluation on training data (1000 cases):

            Decision Tree
          ----------------
          Size      Errors

            72  122(12.2%)   <<


           (a)   (b)    <-classified as
          ----  ----
           206    94    (a): class Bad
            28   672    (b): class Good


        Attribute usage:

        100.00% Status_Of_Existing_Checking_Account
         54.30% Credit_History
         48.40% Duration_In_Month
         35.70% Savings_
```

It can be seen in this example that the error rate has been assessed at 12.2%, suggesting that 87.8% of the time the model correctly classified. A confusion matrix has been written out, however it is more convenient to use the CrossTable function as explained in procedure 100 for the purposes of understanding false positive ratios.

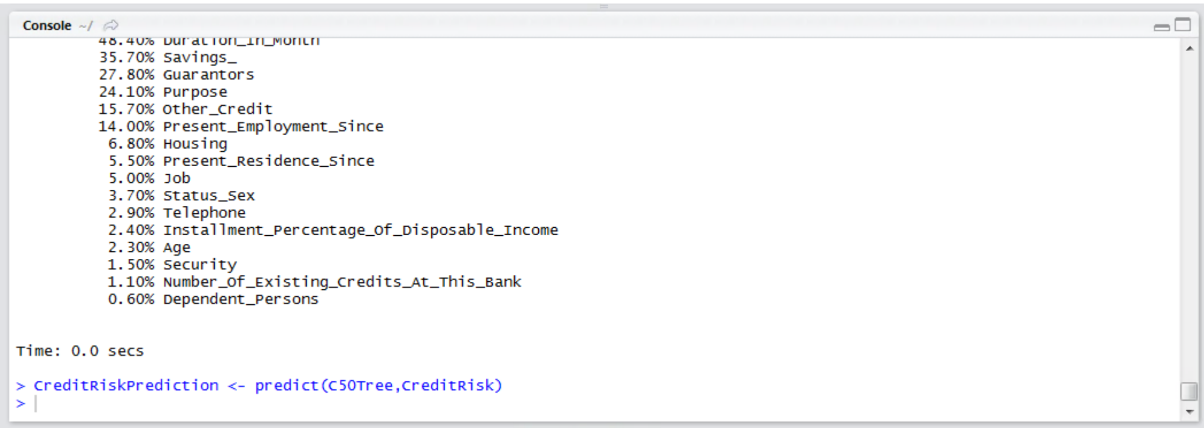
Procedure 5: Recalling a C5 Decision Tree.

As is the case of the majority of models in R, the predict function can take a model object and a data frame as its argument:

CreditRiskPrediction <- predict(C50Tree,CreditRisk)

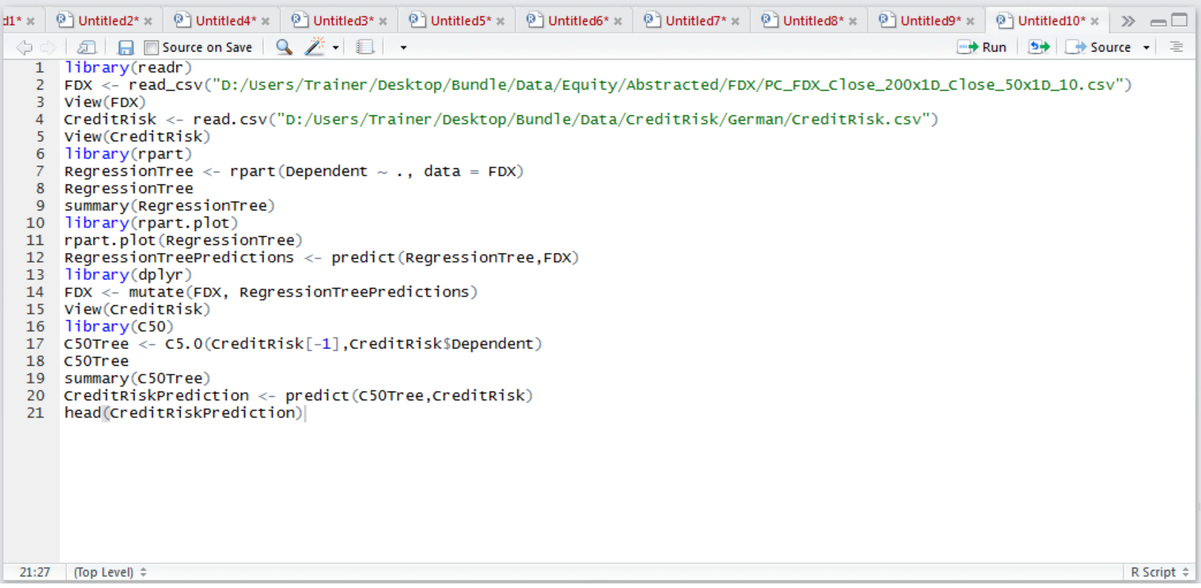


Run the line of script to console:

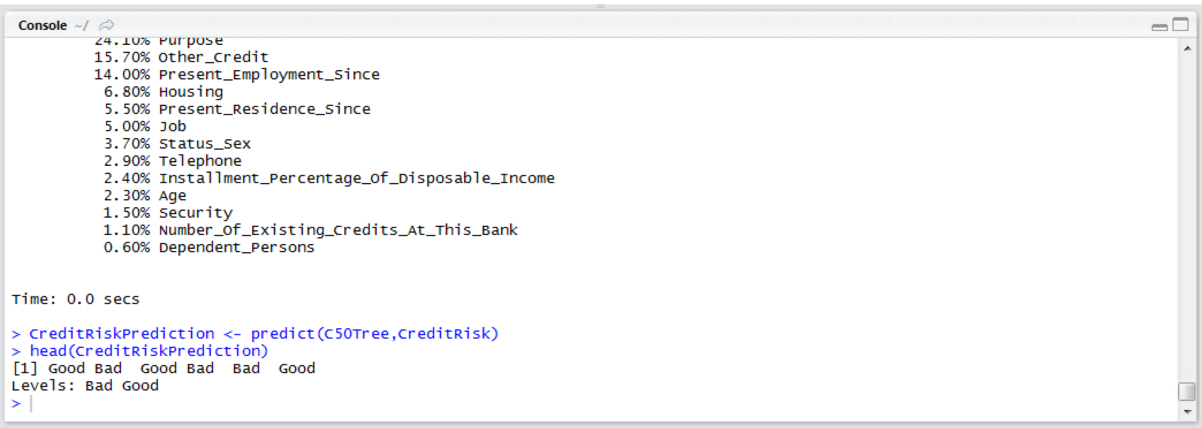


For time being, do not add the vector to the data frame as revised decision tree will be created subsequent to this procedure and owing to the different signature used in training a C5 model, it would be interpreted as an independent variable in its own right. Use the head() function to take a peek at the classification results:

head(CreditRiskPrediction)



Run the line of script to console:



It can be observed that a factor has been created and there are several entries of textual classification result.

Procedure 6: Creating a Confusion Matrix for a C5 Decision Tree.

Beyond the summary statistic created, the confusion matrix is the most convenient means to appraise the utility of a classification model. The confusion matrix for the C5 decision tree model will be created in the same manner as procedure 100:

library("gmodels")

CrossTable(CreditRisk$Dependent, CreditRiskPrediction)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
```

Run the line of script to console:

```
                    | CreditRiskPrediction
CreditRisk$Dependent |       Bad |      Good | Row Total |
--------------------|-----------|-----------|-----------|
                Bad |       206 |        94 |       300 |
                    |   262.701 |    80.251 |           |
                    |     0.687 |     0.313 |     0.300 |
                    |     0.880 |     0.123 |           |
                    |     0.206 |     0.094 |           |
--------------------|-----------|-----------|-----------|
               Good |        28 |       672 |       700 |
                    |   112.586 |    34.393 |           |
                    |     0.040 |     0.960 |     0.700 |
                    |     0.120 |     0.877 |           |
                    |     0.028 |     0.672 |           |
--------------------|-----------|-----------|-----------|
       Column Total |       234 |       766 |      1000 |
                    |     0.234 |     0.766 |           |
--------------------|-----------|-----------|-----------|

>
```

The overall utility of the C5 decision tree model can be inferred in the same manner as procedure 100.

The confusion matrix classified 206 records as being bad correctly, taking CreditRiskPrediction column wise, it can be seen that 28 records were classified as Bad yet they were in fact Good. It can be said that there is an 11.9% error rate on records classified as bad by the model. Taking note of this metric, in procedure 112 boosting will be attempted which should bring about improvement of this model.

Procedure 7: Visualising a C5 Decision Tree.

To visualise a C5 Decision tree, the plot() function from the R base functions can be used, passing the C5 decision tree model as the argument:

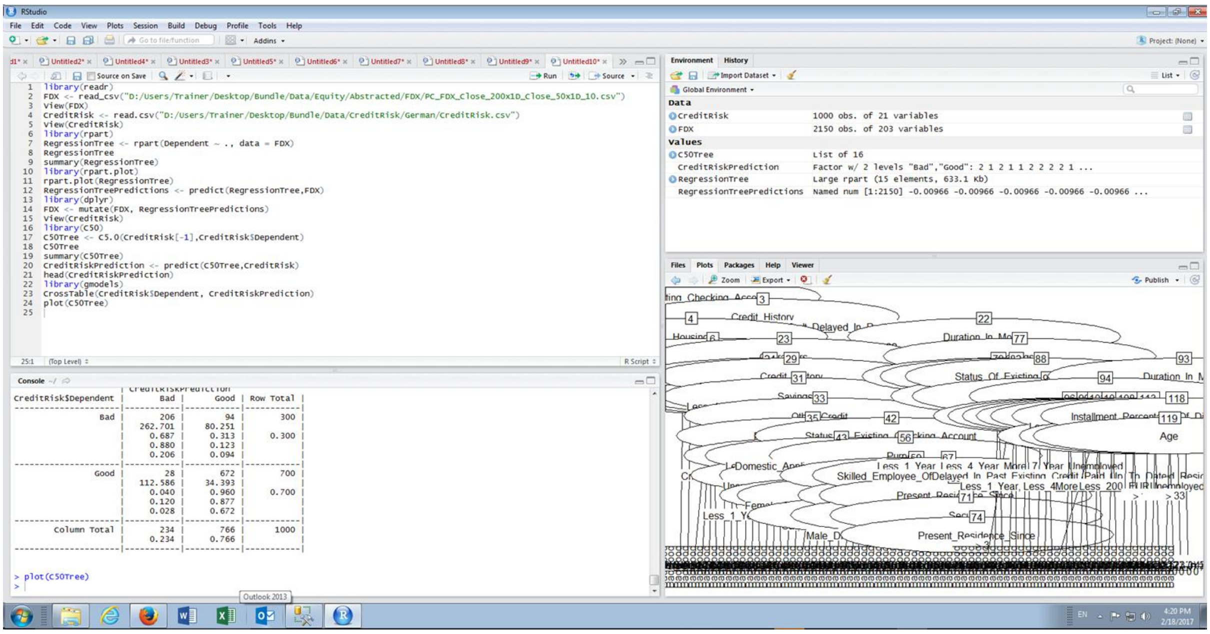
plot(C50Tree)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
```

Run the line of script to console:

```
                     | CreditRiskPrediction
CreditRisk$Dependent |       Bad |      Good | Row Total |
---------------------|-----------|-----------|-----------|
                 Bad |       206 |        94 |       300 |
                     |   262.701 |    80.251 |           |
                     |     0.687 |     0.313 |     0.300 |
                     |     0.880 |     0.123 |           |
                     |     0.206 |     0.094 |           |
---------------------|-----------|-----------|-----------|
                Good |        28 |       672 |       700 |
                     |   112.586 |    34.393 |           |
                     |     0.040 |     0.960 |     0.700 |
                     |     0.120 |     0.877 |           |
                     |     0.028 |     0.672 |           |
---------------------|-----------|-----------|-----------|
        Column Total |       234 |       766 |      1000 |
                     |     0.234 |     0.766 |           |
---------------------|-----------|-----------|-----------|

> plot(C50Tree)
>
```

It can be seen that a visualisation has been written out to the plots pane:



If the tree is very large, then the zoom feature will need to be used to ensure that the plot fits the screen. Even with zoon, it is possibly more appropriate to communicate the product of C5 decision trees as a list of rules, as covered in procedure 111.

Procedure 8: Expressing Business Rules from C5.

In traversing the C5 decision tree it is almost certain that when coming to deploy the model, beyond using the predict() function as described in procedure 108, that it will be expressed or programmed as logical statements, for example:

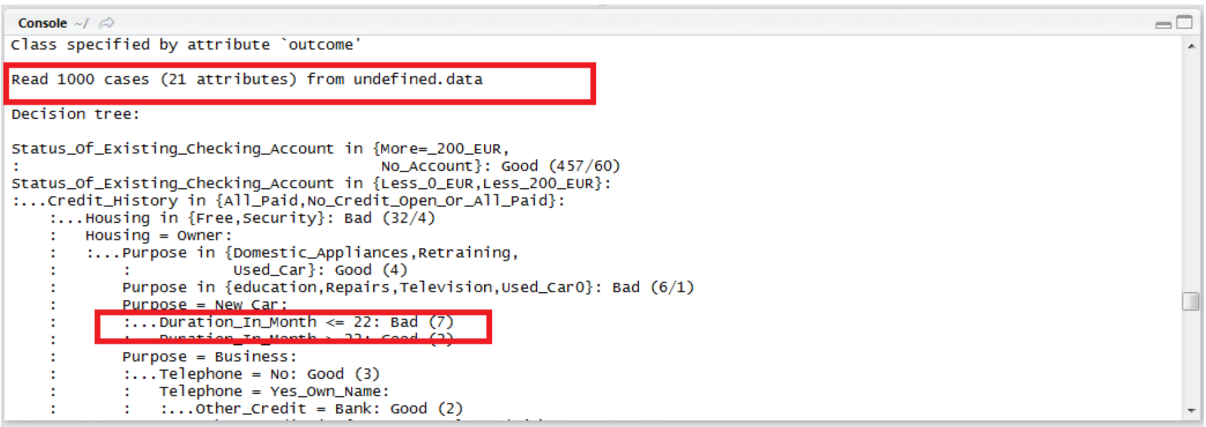
If Status_Of_Existing_Checking_Account < 200 EUR

AND Credit_History in ("All_Paid","No_Credit_Open_Or_All_Paid")

AND Housing = "Owner"

AND Purpose = "New Car"

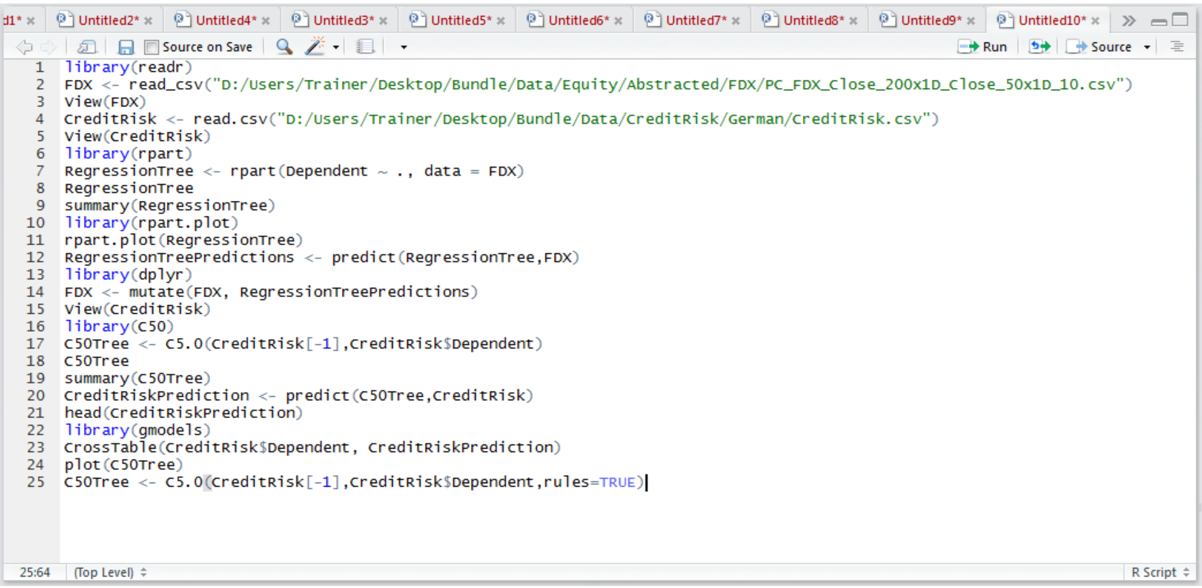
AND Duration_In_Month < 22 THEN "Good"



```
Console ~/
Class specified by attribute `outcome'

Read 1000 cases (21 attributes) from undefined.data

Decision tree:

Status_Of_Existing_Checking_Account in {More=_200_EUR,
:                                        No_Account}: Good (457/60)
Status_Of_Existing_Checking_Account in {Less_0_EUR,Less_200_EUR}:
:...Credit_History in {All_Paid,No_Credit_Open_Or_All_Paid}:
    :...Housing in {Free,Security}: Bad (32/4)
    :   Housing = Owner:
    :   :...Purpose in {Domestic_Appliances,Retraining,
    :       :           Used_Car}: Good (4)
    :       Purpose in {education,Repairs,Television,Used_Car0}: Bad (6/1)
    :       Purpose = New Car:
    :       :...Duration_In_Month <= 22: Bad (7)
    :       :   Duration_In_Month > 22: Good (2)
    :       Purpose = Business:
    :       :...Telephone = No: Good (3)
    :       :   Telephone = Yes_Own_Name:
    :       :   :...Other_Credit = Bank: Good (2)
```

To display the model as rules rather than a tree, it is necessary to rebuild the model specifying rules argument to be true:

C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
```

Thereafter, the summary() function can be used to output a series of rules created in the rebuild as opposed to a decision tree:

summary(C50Tree)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
```

Run the line of script to console:

```
	96.00% Credit_History
	83.50% Status_Of_Existing_Checking_Account
	28.00% Guarantors
	22.10% Savings_
	21.60% Purpose
	17.60% Duration_In_Month
	11.60% Housing
	 7.10% Present_Employment_Since
	 3.80% Present_Residence_Since
	 3.60% Other_Credit
	 3.30% Job
	 2.40% Security
	 2.20% Installment_Percentage_Of_Disposable_Income
	 2.00% Telephone
	 1.50% Status_Sex
	 1.20% Dependent_Persons
	 0.60% Number_Of_Existing_Credits_At_This_Bank


Time: 0.1 secs
>
```

Scrolling up in the console, it can be observed, towards the top, that in place of a decision tree a series of rules has been created:

```
C5.0 [Release 2.07 GPL Edition]    	Sat Feb 18 16:42:34 2017
-------------------------------

Class specified by attribute `outcome'

Read 1000 cases (21 attributes) from undefined.data

Rules:

Rule 1: (12, lift 3.1)
	Status_Of_Existing_Checking_Account in {Less_0_EUR, Less_200_EUR}
	Duration_In_Month > 22
	Purpose = Business
	Savings_ = Less_100_EUR
	Dependent_Persons <= 1
	Telephone = Yes_Own_Name
	->  class Bad  [0.929]

Rule 2: (11, lift 3.1)
	Status_Of_Existing_Checking_Account in {Less_0_EUR, Less_200_EUR}
	Duration_In_Month <= 22
```

These rules can be deployed with very small modification far more intuitively in a variety of languages, not least SQL.

Procedure 9: Boosting and Recalling in C5.

Boosting is a mechanism inside the C5 package that will create many different models, then give opportunity for each model to vote a classification, with the most widely suggested classification being the prevailing classification. The majority classification voted for wins.

In addition to that specified in procedure 107, simply add the argument 10 to indicate that there should be ten trials to vote:

C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
```

Run the line of script to console:

```
	83.50% Status_Of_Existing_Checking_Account
	28.00% Guarantors
	22.10% Savings_
	21.60% Purpose
	17.60% Duration_In_Month
	11.60% Housing
	 7.10% Present_Employment_Since
	 3.80% Present_Residence_Since
	 3.60% Other_Credit
	 3.30% Job
	 2.40% Security
	 2.20% Installment_Percentage_Of_Disposable_Income
	 2.00% Telephone
	 1.50% Status_Sex
	 1.20% Dependent_Persons
	 0.60% Number_Of_Existing_Credits_At_This_Bank


Time: 0.1 secs

> C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
> 
```

The summary function will produce a report in a similar manner to that observed in procedure 107:

summary(C50Tree)

```
Console ~/
        100.00% Purpose
        100.00% Foreign_Worker
         99.20% Guarantors
         97.50% Savings_
         95.00% Other_Credit
         92.40% Housing
         91.10% Security
         91.00% Present_Employment_Since
         90.40% Requested_Amount
         85.40% Job
         74.50% Age
         70.60% Installment_Percentage_Of_Disposable_Income
         68.00% Present_Residence_Since
         58.10% Number_Of_Existing_Credits_At_This_Bank
         56.10% Telephone
         55.00% Status_Sex
         52.40% Dependent_Persons


Time: 0.1 secs

>
```

In this instance, however, upon scrolling up, it can be seen that several different models \ trials have been created:

```
Console ~/

SubTree [S5]

Installment_Percentage_Of_Disposable_Income <= 2: Good (9.4/3.6)
Installment_Percentage_Of_Disposable_Income > 2: Bad (23.3/1.6)

-----  Trial 9:  -----

Decision tree:

Status_Of_Existing_Checking_Account = No_Account:
:...Other_Credit in {Bank,Stores}:
:   :...Guarantors in {Guarantor,Joint}: Good (3)
:   :   Guarantors = None:
:   :   :...Installment_Percentage_Of_Disposable_Income <= 1: Good (7.1/0.5)
:   :       Installment_Percentage_Of_Disposable_Income > 1:
:   :       :...Purpose in {Domestic_Appliances,Repairs,Retraining,
:   :           :              Used_Car0}: Bad (0)
:   :           Purpose in {Furniture,Television}: Good (30.4/7.1)
:   :           Purpose in {Business,education,New_Car,Used_Car}:
:   :           :...Duration_In_Month <= 9: Good (2.4)
:   :               Duration_In_Month > 9:
```

In the above example the decision tree for the 9th trial has been evidenced. Prediction takes place in exactly the same manner, using the predict() function, except for it will run several models and established a voted majority classification. This is boosting:

CreditRiskPrediction <- predict(C50Tree,CreditRisk)

```
View(CreditRisk)
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
```

Run the line of script to console:

```
Console ~/
       100.00% Foreign_worker
        99.20% Guarantors
        97.50% Savings_
        95.00% Other_Credit
        92.40% Housing
        91.10% Security
        91.00% Present_Employment_Since
        90.40% Requested_Amount
        85.40% Job
        74.50% Age
        70.60% Installment_Percentage_Of_Disposable_Income
        68.00% Present_Residence_Since
        58.10% Number_Of_Existing_Credits_At_This_Bank
        56.10% Telephone
        55.00% Status_Sex
        52.40% Dependent_Persons


Time: 0.1 secs

> CreditRiskPrediction <- predict(C50Tree,CreditRisk)
> 
```

A confusion matrix can be created to compare this object with that created in procedure 100:

CrossTable(CreditRisk$Dependent, CreditRiskPrediction)

```
library(rpart)
RegressionTree <- rpart(Dependent ~ ., data = FDX)
RegressionTree
summary(RegressionTree)
library(rpart.plot)
rpart.plot(RegressionTree)
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
```

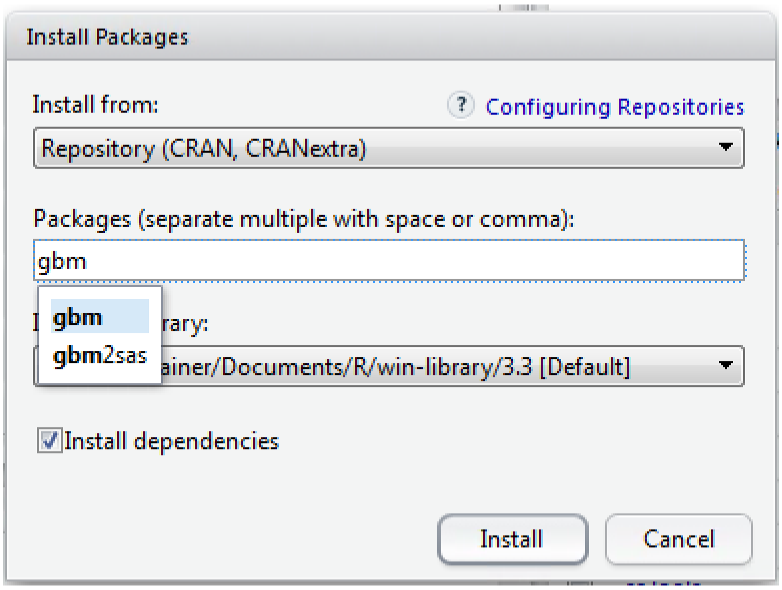
Run the line of script to console:

```
Console ~/
                    | CreditRiskPrediction 
CreditRisk$Dependent |       Bad |      Good | Row Total | 
--------------------|-----------|-----------|-----------|
                Bad |       280 |        20 |       300 | 
                    |   454.312 |   177.554 |           | 
                    |     0.933 |     0.067 |     0.300 | 
                    |     0.996 |     0.028 |           | 
                    |     0.280 |     0.020 |           | 
--------------------|-----------|-----------|-----------|
               Good |         1 |       699 |       700 | 
                    |   194.705 |    76.095 |           | 
                    |     0.001 |     0.999 |     0.700 | 
                    |     0.004 |     0.972 |           | 
                    |     0.001 |     0.699 |           | 
--------------------|-----------|-----------|-----------|
       Column Total |       281 |       719 |      1000 | 
                    |     0.281 |     0.719 |           | 
--------------------|-----------|-----------|-----------|

> 
```

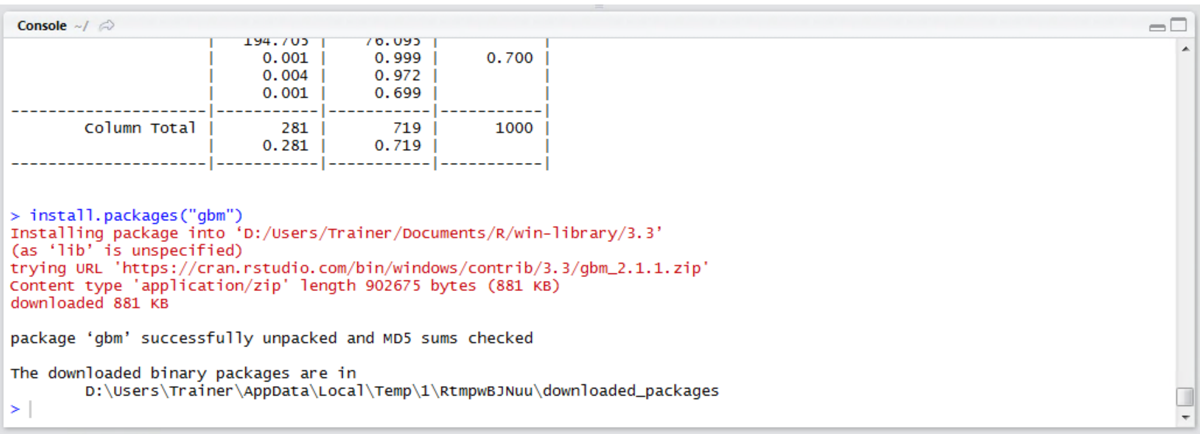
In this example, it can be observed that there were 281 accounts where predicted to be bad, taking the CreditRiskPrediction column-wise, it can be observed there was a 1 account classification as bad in error. Out of 281 classifications as bad, it can be said that the error rate is just 0.3%. Referring to the original model as created in procedure 107, it can be seen that an 11% increase in performance has been achieved from boosting.

Procedure 10: Creating a Gradient Boosting Machine.

A relatively underutilised classification tool, which is built upon the concept of boosted decision trees, is the Gradient Boosting Machine, or GBM. The GBM is a fairly black box implementation of the methods covered thus far, in this module. The concept of Boosting refers to taking underperforming classifications and singling them out for boosting, or rather creating a dedicated model targeting the weaker performing data. The GBM is part of the GBM package, as such install that package:

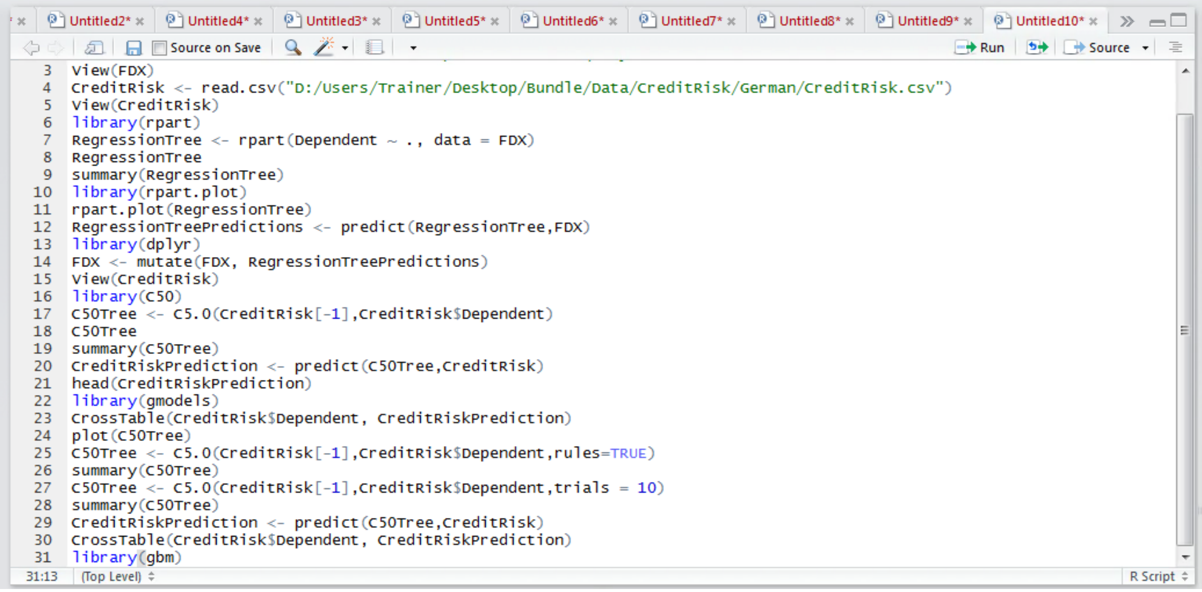


Click Install to download and install the package:

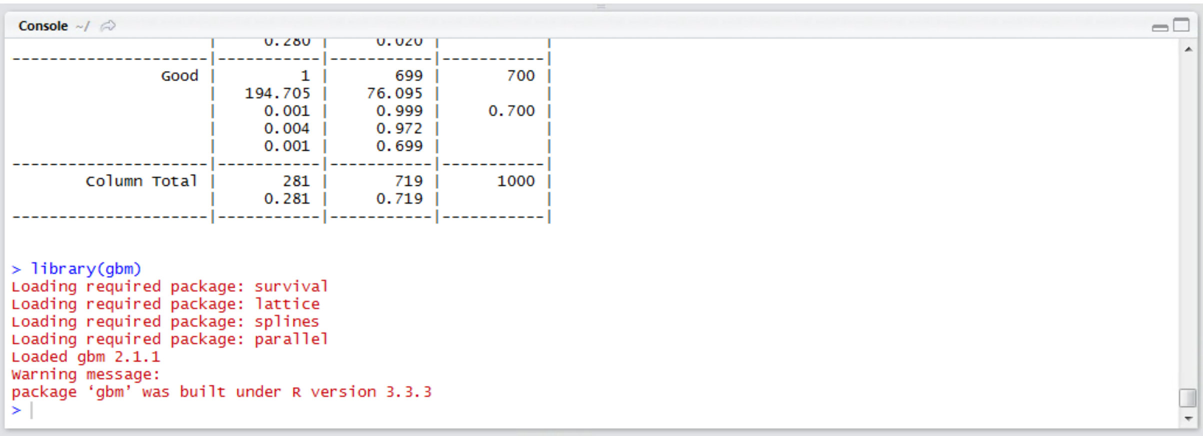


Load the library:

library(GBM)



Run the line of script to console:



The warning messages can be ignored as we can be reasonably assured of backward compatibility between the package build and this version of R.

Creating a GBM is similar to the familiar interfaces of regression, except for having a few parameters relating to the taming of the GBM:

```
gbm = gbm(Dependent ~., CreditRisk,
    n.trees=1000,
    shrinkage=0.01,
    distribution="gaussian",
    interaction.depth=7,
    bag.fraction=0.9,
    cv.fold=10,
    n.minobsinnode = 50
)
```

Run the line of script to console:

```
RegressionTreePredictions <- predict(RegressionTree,FDX)
library(dplyr)
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
```

Run the line of script to console, it may take some time:

```
                     |     0.281 |     0.719 |           |
---------------------|-----------|-----------|-----------|

> library(gbm)
Loading required package: survival
Loading required package: lattice
Loading required package: splines
Loading required package: parallel
Loaded gbm 2.1.1
Warning message:
package ‘gbm’ was built under R version 3.3.3
> gbm = gbm(Dependent ~., CreditRisk,
+           n.trees=1000,
+           shrinkage=0.01,
+           distribution="gaussian",
+           interaction.depth=7,
+           bag.fraction=0.9,
+           cv.fold=10,
+           n.minobsinnode = 50
+ )
>
```

To review the performance statistics of the GBM, simply recall the model:

gbm

```
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
```

Run the line of script to console:

```
Loading required package: parallel
Loaded gbm 2.1.1
Warning message:
package ‘gbm’ was built under R version 3.3.3
> gbm = gbm(Dependent ~., CreditRisk,
+           n.trees=1000,
+           shrinkage=0.01,
+           distribution="gaussian",
+           interaction.depth=7,
+           bag.fraction=0.9,
+           cv.fold=10,
+           n.minobsinnode = 50
+ )
> gbm
gbm(formula = Dependent ~ ., distribution = "gaussian", data = CreditRisk,
    n.trees = 1000, interaction.depth = 7, n.minobsinnode = 50,
    shrinkage = 0.01, bag.fraction = 0.9, cv.folds = 10)
A gradient boosted model with gaussian loss function.
1000 iterations were performed.
The best cross-validation iteration was 542.
There were 20 predictors of which 19 had non-zero influence.
>
```

The most salient information from this summary is that 1000 iterations were performed, with the cross validation diverging at tree 542. A visual inspection of the cross validation can be presented by:

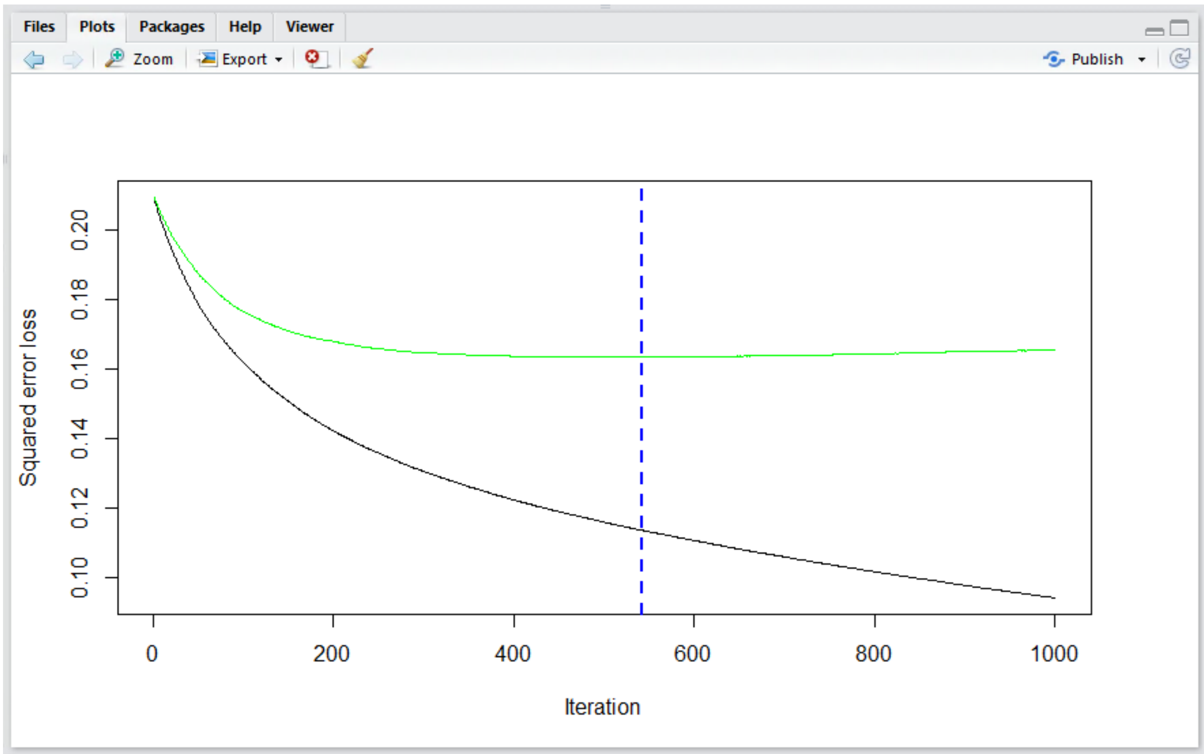
gbm.perf(gbm)

```
FDX <- mutate(FDX, RegressionTreePredictions)
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
gbm.perf(gbm)
```

Run the line of script to console:

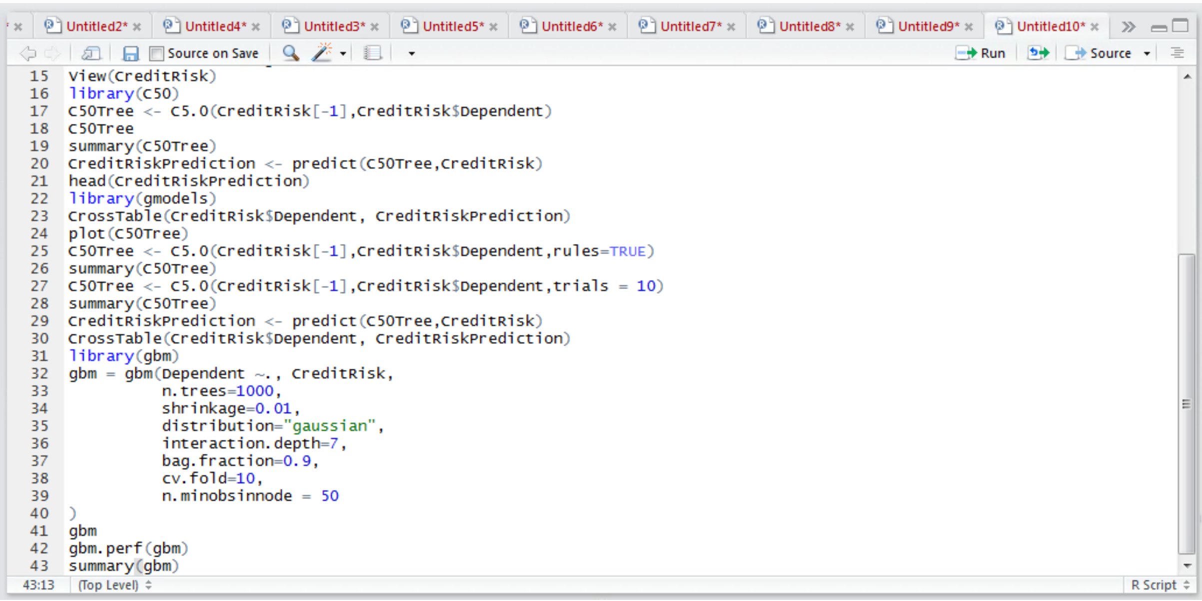
```
package 'gbm' was built under R version 3.3.3
> gbm = gbm(Dependent ~., CreditRisk,
+           n.trees=1000,
+           shrinkage=0.01,
+           distribution="gaussian",
+           interaction.depth=7,
+           bag.fraction=0.9,
+           cv.fold=10,
+           n.minobsinnode = 50
+ )
> gbm
gbm(formula = Dependent ~ ., distribution = "gaussian", data = CreditRisk,
    n.trees = 1000, interaction.depth = 7, n.minobsinnode = 50,
    shrinkage = 0.01, bag.fraction = 0.9, cv.folds = 10)
A gradient boosted model with gaussian loss function.
1000 iterations were performed.
The best cross-validation iteration was 542.
There were 20 predictors of which 19 had non-zero influence.
> gbm.perf(gbm)
Using cv method...
[1] 542
>
```

It can be seen that the line was drawn at the point divergence started:



As decision trees can become a little unwieldy, it might be prudent to inspect the relative importance of each of the independent variables with a view to pruning and rerunning the GBM training. To understand the importance of each Independent Variable, wrap the summary function around the GBM:

summary(GBM)

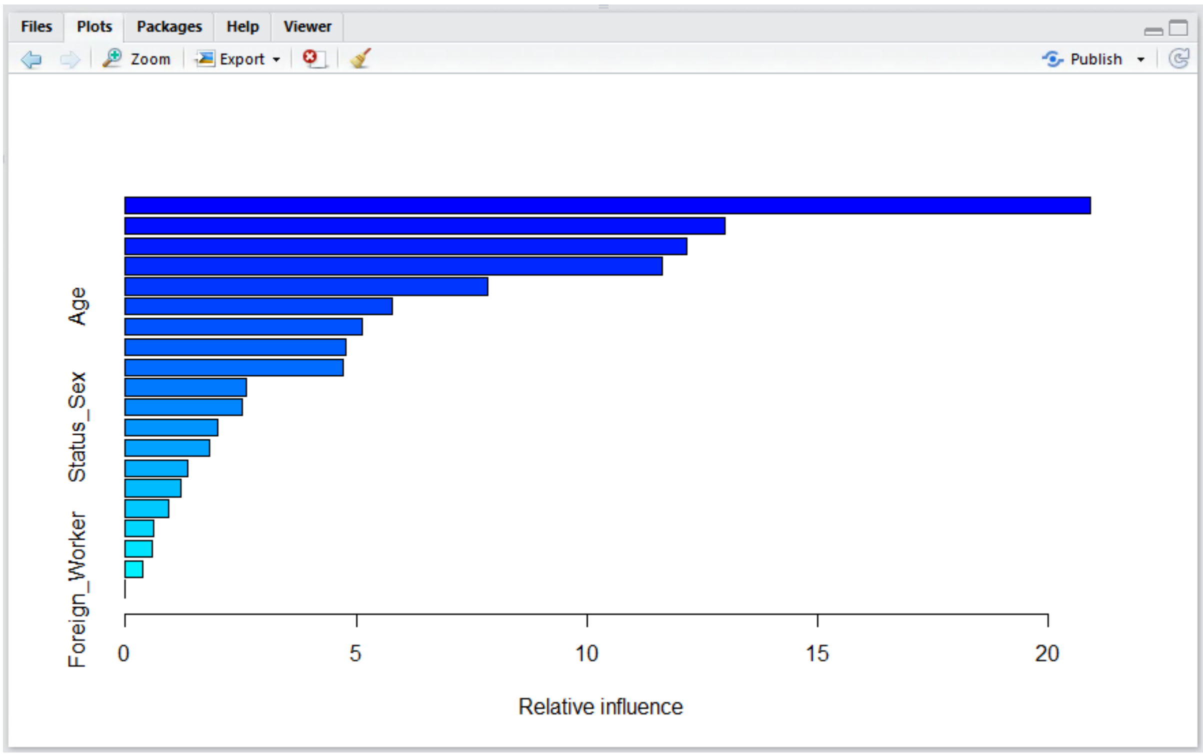


```
View(CreditRisk)
library(C50)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
gbm.perf(gbm)
summary(gbm)
```

Run the line of script to console:

```
Console ~/
                                                                       var    rel.inf
Status_Of_Existing_Checking_Account               Status_Of_Existing_Checking_Account 20.8972034
Requested_Amount                                                     Requested_Amount 12.9943387
Purpose                                                                       Purpose 12.1566344
Duration_In_Month                                                   Duration_In_Month 11.6283011
Credit_History                                                         Credit_History  7.8556274
Age                                                                               Age  5.7947571
Savings_                                                                     Savings_  5.1375370
Present_Employment_Since                                     Present_Employment_Since  4.7706571
Security                                                                     Security  4.7114077
Installment_Percentage_Of_Disposable_Income Installment_Percentage_Of_Disposable_Income  2.6239241
Other_Credit                                                             Other_Credit  2.5359300
Status_Sex                                                                 Status_Sex  2.0092849
Present_Residence_Since                                       Present_Residence_Since  1.8121788
Housing                                                                       Housing  1.3485600
Job                                                                               Job  1.2072516
Telephone                                                                   Telephone  0.9482596
Number_Of_Existing_Credits_At_This_Bank       Number_Of_Existing_Credits_At_This_Bank  0.6025786
Guarantors                                                                 Guarantors  0.5825072
Dependent_Persons                                                   Dependent_Persons  0.3830613
Foreign_Worker                                                         Foreign_Worker  0.0000000
>
```

The most useful and important variable is written out first, with the less important being written out last. This is also displayed in a bar chart giving the overall usefulness of the independent variables at a glance:



Procedure 11: Recalling a Gradient Boosting Machine.

Recalling the GBM is quite initiative and obeys the standardised predict signature. To recall the GBM:

GBMPredictions <- predict(GBM,CreditRisk,type = "response")

```
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)

library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
gbm.perf(gbm)
summary(gbm)
GBMPredictions <- predict(gbm,CreditRisk,type = "response")

```

Run the line of script to console:

```
Requested_Amount                                                   Requested_Amount 12.7710491
Purpose                                                                     Purpose 12.3650978
Duration_In_Month                                                 Duration_In_Month 11.9321144
Credit_History                                                       Credit_History  7.6189051
Age                                                                             Age  5.7622241
Savings_                                                                   Savings_  5.3342739
Present_Employment_Since                                   Present_Employment_Since  4.8073478
Security                                                                   Security  4.5046783
Installment_Percentage_of_Disposable_Income Installment_Percentage_of_Disposable_Income  2.4276679
Other_Credit                                                           Other_Credit  2.3537318
Present_Residence_Since                                     Present_Residence_Since  2.1845024
Status_Sex                                                               Status_Sex  2.0042821
Job                                                                             Job  1.2240959
Housing                                                                     Housing  1.1816382
Telephone                                                                 Telephone  1.0844734
Number_of_Existing_Credits_At_This_Bank     Number_of_Existing_Credits_At_This_Bank  0.6411613
Guarantors                                                               Guarantors  0.6411468
Dependent_Persons                                                 Dependent_Persons  0.3708990
Foreign_Worker                                                       Foreign_Worker  0.0000000
> GBMPredictions <- predict(gbm,CreditRisk,type = "response")
Using 494 trees...
> 
```

A distinct peculiarity, given that the CreditRisk data frame has a dependent variable which is a factor, is that the binary classification has been modelled between 1 and 2, being the levels of the factor with 1 being Bad, and Good being two:

```
Data
CreditRisk         1000 obs. of 21 variables
  Dependent : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
  Status_of_Existing_Checking_Account : Factor w/ 4 levels "Less_0_EUR","Less_200_EUR",..: 1 2 4 1 1 4 ...
  Duration_In_Month : int 6 48 12 42 24 36 24 36 12 30 ...
  Credit_History : Factor w/ 5 levels "All_Paid","Critical_Account_Default",..: 2 4 2 4 3 4 4 4 2 ...
  Purpose : Factor w/ 10 levels "Business","Domestic_Appliances",..: 8 8 3 4 5 3 4 9 8 5 ...
  Requested_Amount : int 1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
  Savings_ : Factor w/ 5 levels "Less_100_EUR",..: 5 1 1 1 1 5 2 1 4 1 ...
  Present_Employment_Since : Factor w/ 5 levels "Less_1_Year",..: 4 2 3 3 2 2 4 2 3 5 ...
  Installment_Percentage_of_Disposable_Income: int 4 2 2 2 3 2 3 2 2 4 ...
  Status_Sex : Factor w/ 4 levels "Female_Divorced_Separated",..: 4 1 4 4 4 4 4 4 2 3 ...
  Guarantors : Factor w/ 3 levels "Guarantor","Joint",..: 3 3 3 1 3 3 3 3 3 3 ...
  Present_Residence_Since : int 4 2 3 4 4 4 4 2 4 2 ...
```

It follows that predictions that are closer to 2, than 1 would be considered to be Good, whereas vice versa, 1. To appraise the model performance, a confusion matrix should be created. Create a vector using the ifelse() function to classify between Good and Bad:

CreditRiskGBMClassifications <- ifelse(GBMPredictions >= 1.5,"Good","Bad")

```
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent)
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
gbm.perf(gbm)
summary(gbm)
GBMPredictions <- predict(gbm,CreditRisk,type = "response")
CreditRiskGBMClassifications <- ifelse(GBMPredictions >= 1.5,"Good","Bad")
```

Run the line of script to console:

```
Purpose                                                                   Purpose 12.3650978
Duration_In_Month                                               Duration_In_Month 11.9321144
Credit_History                                                     Credit_History  7.6189051
Age                                                                           Age  5.7622241
Savings_                                                                 Savings_  5.3342739
Present_Employment_Since                                 Present_Employment_Since  4.8073478
Security                                                                 Security  4.5046783
Installment_Percentage_Of_Disposable_Income Installment_Percentage_Of_Disposable_Income  2.4276679
Other_Credit                                                         Other_Credit  2.3537318
Present_Residence_Since                                   Present_Residence_Since  2.1845024
Status_Sex                                                             Status_Sex  2.0042821
Job                                                                           Job  1.2240959
Housing                                                                   Housing  1.1816382
Telephone                                                               Telephone  1.0844734
Number_Of_Existing_Credits_At_This_Bank         Number_Of_Existing_Credits_At_This_Bank  0.6411613
Guarantors                                                             Guarantors  0.6411468
Dependent_Persons                                               Dependent_Persons  0.3708990
Foreign_Worker                                                     Foreign_Worker  0.0000000
> GBMPredictions <- predict(gbm,CreditRisk,type = "response")
Using 494 trees...
> CreditRiskGBMClassifications <- ifelse(GBMPredictions >= 1.5,"Good","Bad")
>
```

Create a confusion matrix between the actual value and the value predicted by the GBM:

CrossTable(CreditRisk$Dependent, CreditRiskGBMClassifications)

```r
C50Tree
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
head(CreditRiskPrediction)
library(gmodels)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
plot(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,rules=TRUE)
summary(C50Tree)
C50Tree <- C5.0(CreditRisk[-1],CreditRisk$Dependent,trials = 10)
summary(C50Tree)
CreditRiskPrediction <- predict(C50Tree,CreditRisk)
CrossTable(CreditRisk$Dependent, CreditRiskPrediction)
library(gbm)
gbm = gbm(Dependent ~., CreditRisk,
          n.trees=1000,
          shrinkage=0.01,
          distribution="gaussian",
          interaction.depth=7,
          bag.fraction=0.9,
          cv.fold=10,
          n.minobsinnode = 50
)
gbm
gbm.perf(gbm)
summary(gbm)
GBMPredictions <- predict(gbm,CreditRisk,type = "response")
CreditRiskGBMClassifications <- ifelse(GBMPredictions >= 1.5,"Good","Bad")
CrossTable(CreditRisk$Dependent, CreditRiskGBMClassifications)
```

Run the line of script to console:

```
                     | CreditRiskGBMClassifications
CreditRisk$Dependent |       Bad |      Good | Row Total |
---------------------|-----------|-----------|-----------|
                 Bad |       182 |       118 |       300 |
                     |   203.879 |    57.504 |           |
                     |     0.607 |     0.393 |     0.300 |
                     |     0.827 |     0.151 |           |
                     |     0.182 |     0.118 |           |
---------------------|-----------|-----------|-----------|
                Good |        38 |       662 |       700 |
                     |    87.377 |    24.645 |           |
                     |     0.054 |     0.946 |     0.700 |
                     |     0.173 |     0.849 |           |
                     |     0.038 |     0.662 |           |
---------------------|-----------|-----------|-----------|
        Column Total |       220 |       780 |      1000 |
                     |     0.220 |     0.780 |           |
---------------------|-----------|-----------|-----------|

>
```

It can be seen in this example that the GBM has mustered a strong performance. Of 220 accounts that were bad, it can be seen that the GBM classified 182 of them correctly, which gives an overall accuracy rating of 82%.

Module 11: Naive Bayesian Classifiers and Laplace Estimator.

A Naive Bayesian Classifier is an extremely powerful general issue classifier that performs well for most classification problems. In addition to providing a predicted classification, it also provides a probability of that classification making it both intuitive and accurate for risk based approaches.

The dataset to be used in this module is the CreditRisk dataset used in module 7, however some consideration needs to be given to the fact that this is contains come continuous data which is not, by default, appropriate for Bayesian analysis, as Bayesian analysis is a question of probability.

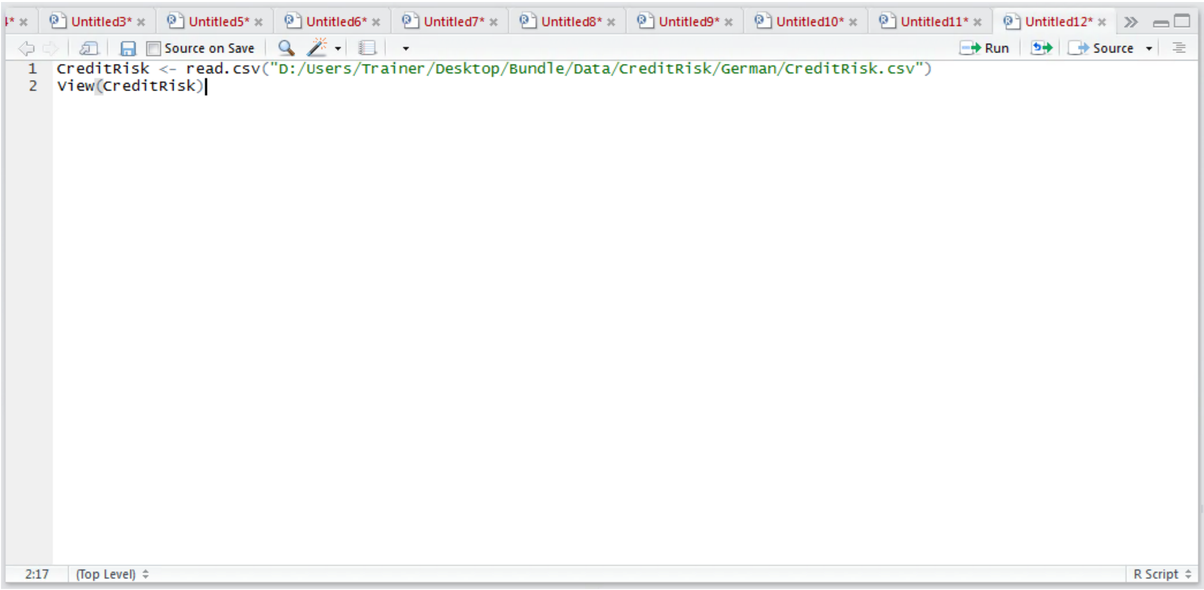
While it is clearly simpler, for the purposes of these procedures, to provide a clean dataset it allows for the introduction of some more advanced data frame manipulation techniques and cements that notion that continuous data is not appropriate for this modelling tool.

Procedure 1: Converting Continuous Data to Categorical Data.

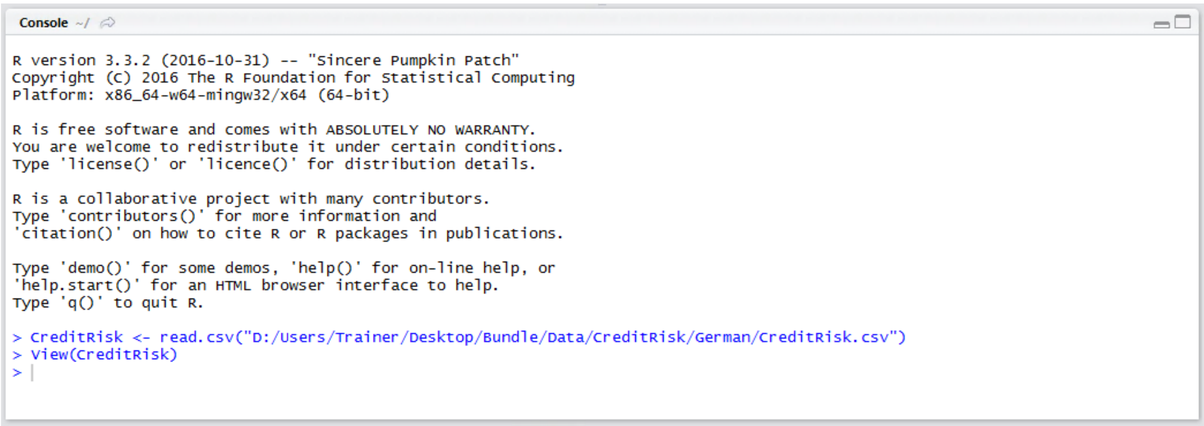
Start by loading the CreditRisk dataset using the base read.csv() function, to assure that strings are converted to factors.

CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")

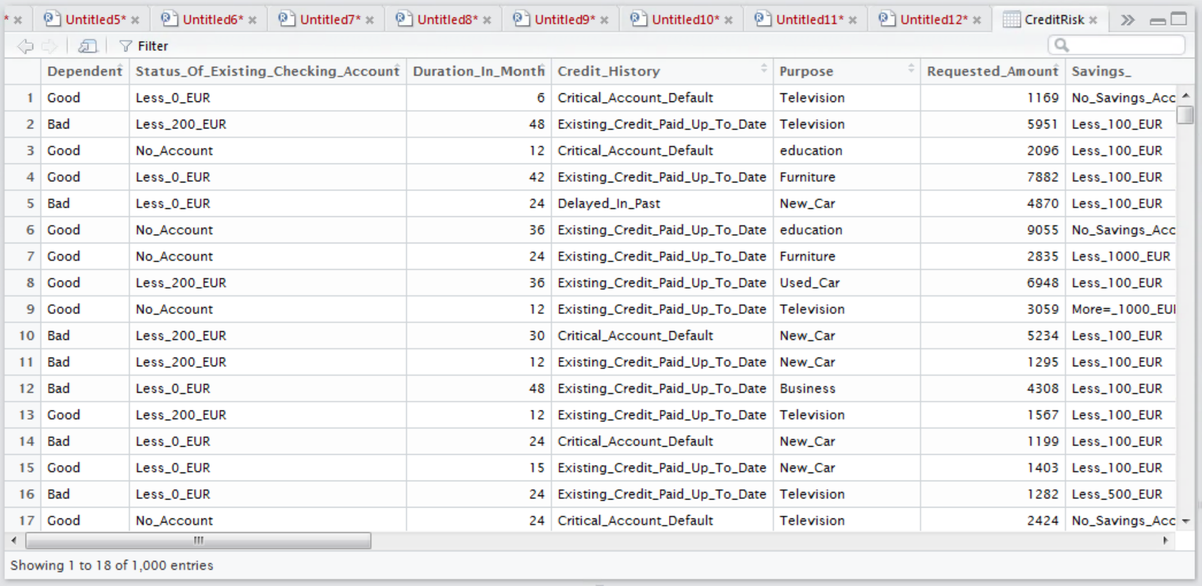
View(CreditRisk)



Run the block of script to console:



The View() function will load the dataset in the R Studio Viewer:



	Dependent	Status_Of_Existing_Checking_Account	Duration_In_Month	Credit_History	Purpose	Requested_Amount	Savings_
1	Good	Less_0_EUR	6	Critical_Account_Default	Television	1169	No_Savings_Acc
2	Bad	Less_200_EUR	48	Existing_Credit_Paid_Up_To_Date	Television	5951	Less_100_EUR
3	Good	No_Account	12	Critical_Account_Default	education	2096	Less_100_EUR
4	Good	Less_0_EUR	42	Existing_Credit_Paid_Up_To_Date	Furniture	7882	Less_100_EUR
5	Bad	Less_0_EUR	24	Delayed_In_Past	New_Car	4870	Less_100_EUR
6	Good	No_Account	36	Existing_Credit_Paid_Up_To_Date	education	9055	No_Savings_Acc
7	Good	No_Account	24	Existing_Credit_Paid_Up_To_Date	Furniture	2835	Less_1000_EUR
8	Good	Less_200_EUR	36	Existing_Credit_Paid_Up_To_Date	Used_Car	6948	Less_100_EUR
9	Good	No_Account	12	Existing_Credit_Paid_Up_To_Date	Television	3059	More=_1000_EU
10	Bad	Less_200_EUR	30	Critical_Account_Default	New_Car	5234	Less_100_EUR
11	Bad	Less_200_EUR	12	Existing_Credit_Paid_Up_To_Date	New_Car	1295	Less_100_EUR
12	Bad	Less_0_EUR	48	Existing_Credit_Paid_Up_To_Date	Business	4308	Less_100_EUR
13	Good	Less_200_EUR	12	Existing_Credit_Paid_Up_To_Date	Television	1567	Less_100_EUR
14	Bad	Less_0_EUR	24	Critical_Account_Default	New_Car	1199	Less_100_EUR
15	Good	Less_0_EUR	15	Existing_Credit_Paid_Up_To_Date	New_Car	1403	Less_100_EUR
16	Bad	Less_0_EUR	24	Existing_Credit_Paid_Up_To_Date	Television	1282	Less_500_EUR
17	Good	No_Account	24	Critical_Account_Default	Television	2424	No_Savings_Acc

Showing 1 to 18 of 1,000 entries

There are several vectors that are not appropriate for Bayesian analysis as they are continuous:

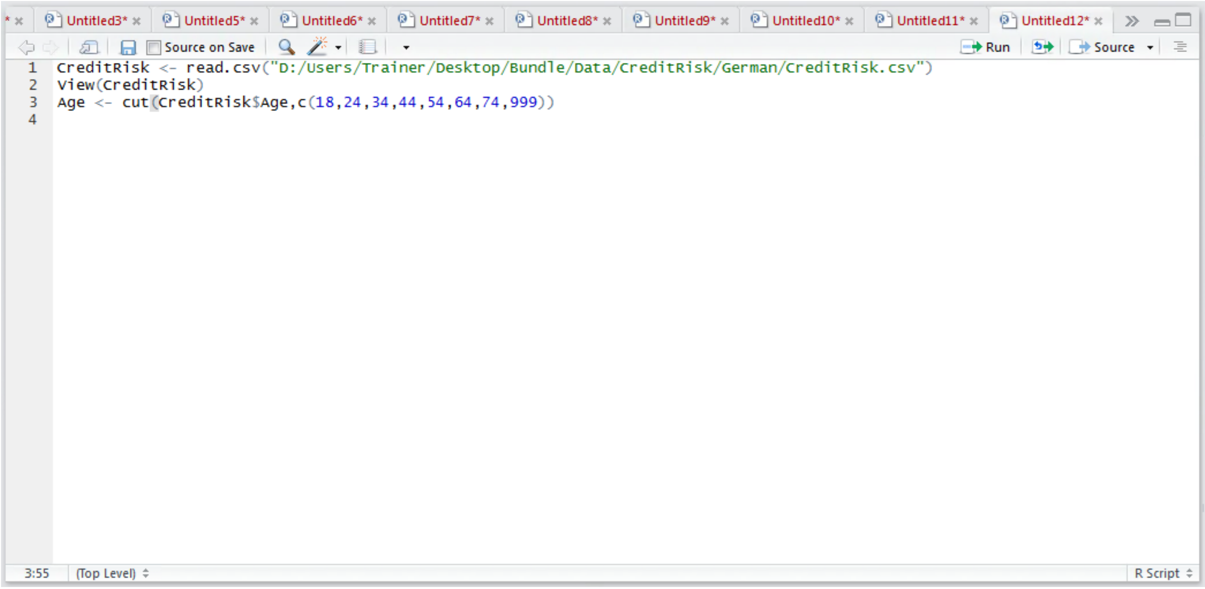
- Requested_Amount.
- Installment_Percentage_Of_Disposable_Income.
- Present_Residency_Since.
- Age.
- Number_Of_Existing_Credits_At_This_Bank.
- Dependent_Persons.

There are a variety of ways to convert the continuous values to categorical data, yet in this example we will focus on binning on a single vector, Age. In this example, the Age will be broken into commonly used Age brackets:

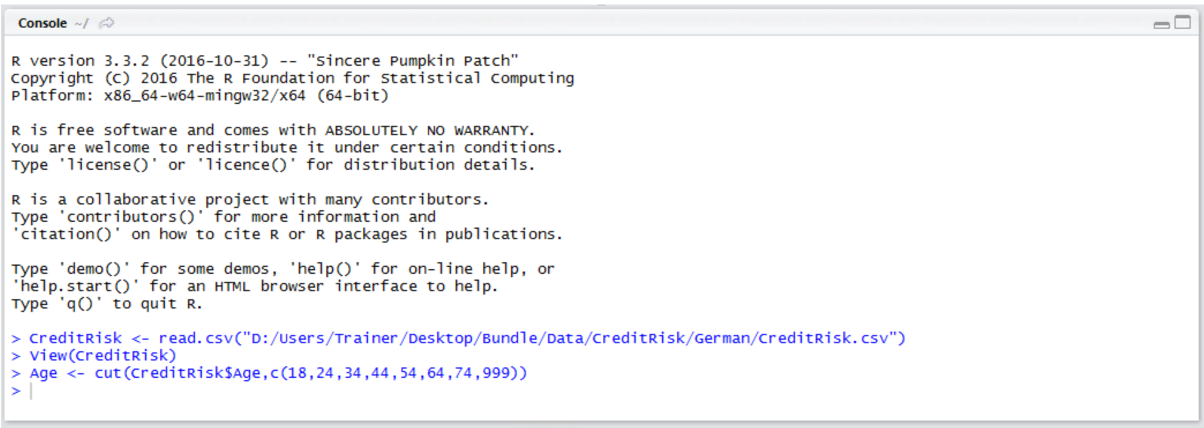
- 18-24 Years old.
- 25-34 Years old.
- 35-44 Years old.
- 45-54 Years old.
- 55-64 Years old.
- 65-74 Years old.
- 75 Years or older.

It would be possible to use a series of logical statements to make the slice, or cut, between the values in this continuous series of data, but it would be quite cumbersome. Fortunately there is a function that can simplify this for us, the cut() function. The cut function takes a vector of data, and a vector of points to make the cut, returning a string denoting the range. To make the cut based on the ranges described:

Age <- cut(CreditRisk$cut,c(18,24,34,44,54,64,74,999))

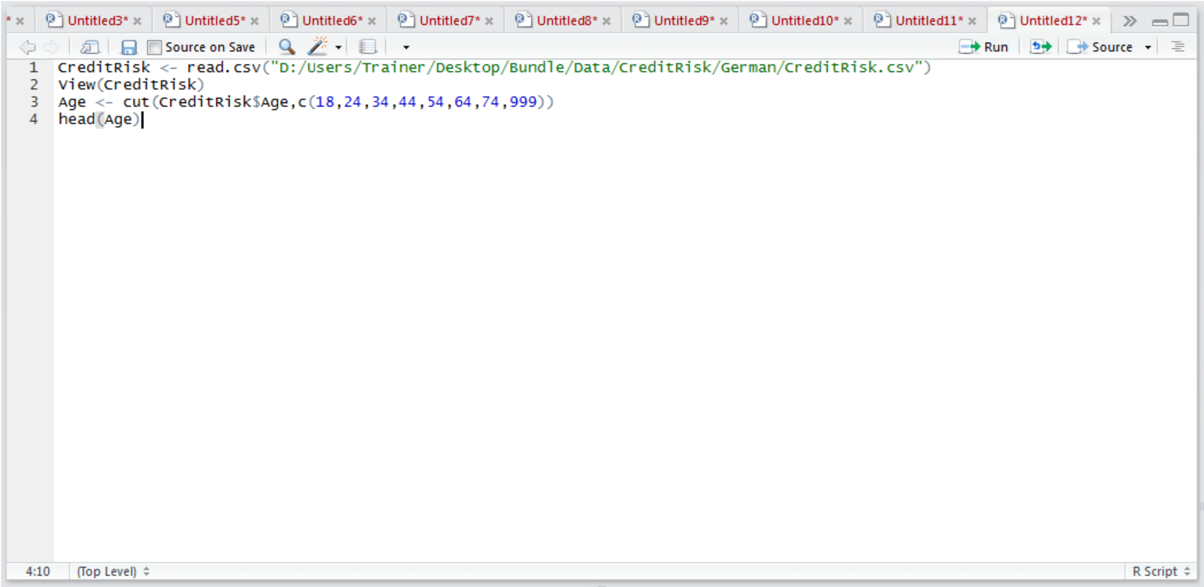


Run the line of script to console:

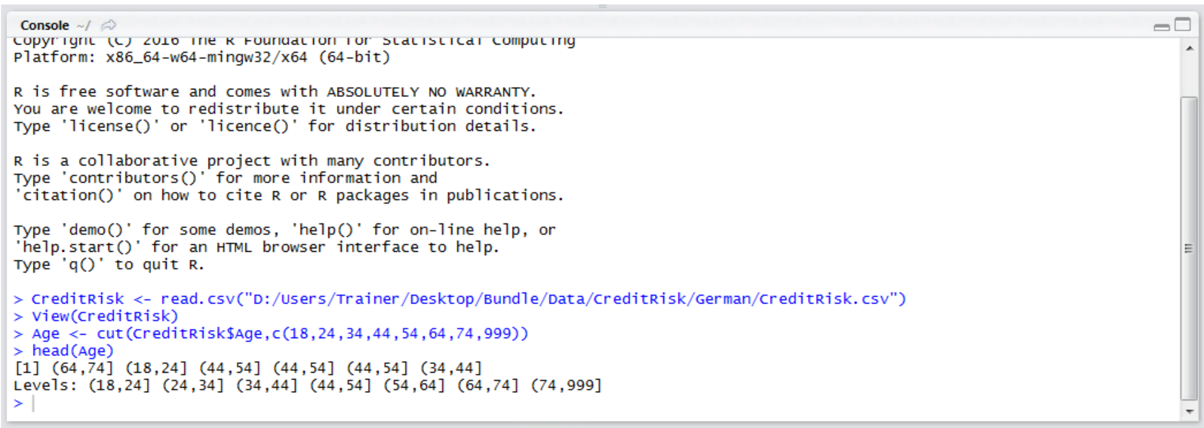


The head() command can used on Age to confirm that it is indeed a factor and that the levels have been apportioned:

head(Age)

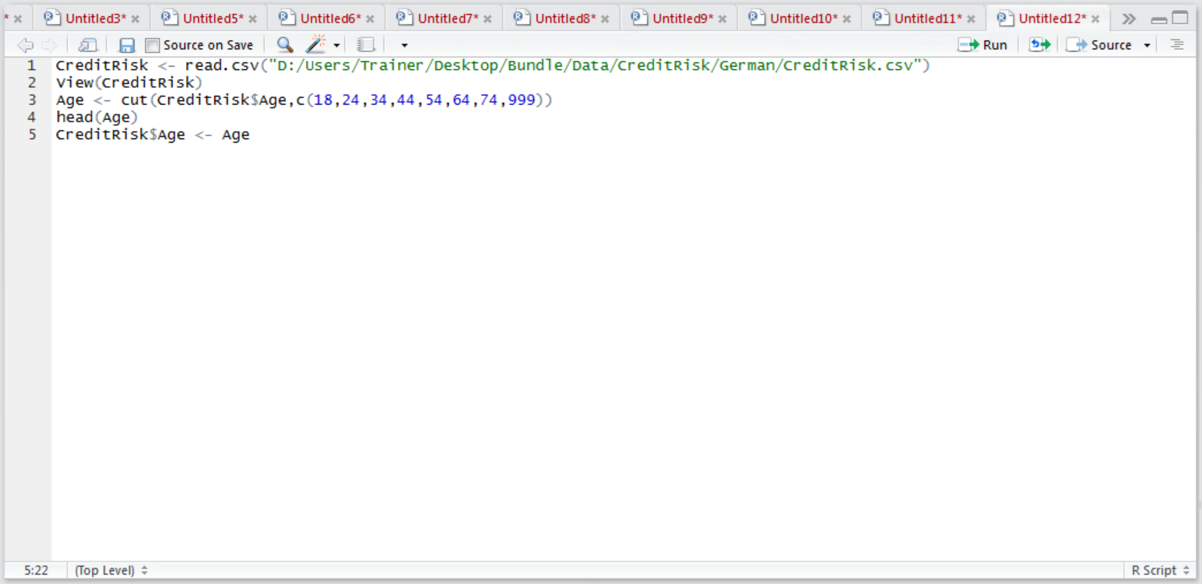


Run the line of script to console:

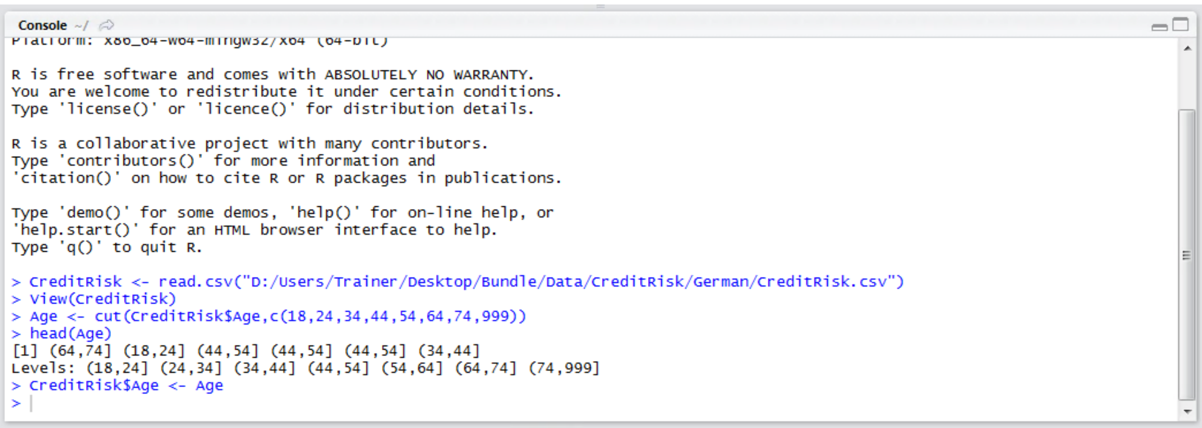


Having created a factor for Age, it is necessary to overwrite the vector in the CreditRisk Data Frame. This is a simple procedure of targeting the Age vector in the data frame as the target of assignment for the Age factor:

```
CreditRisk$Age <- Age
```

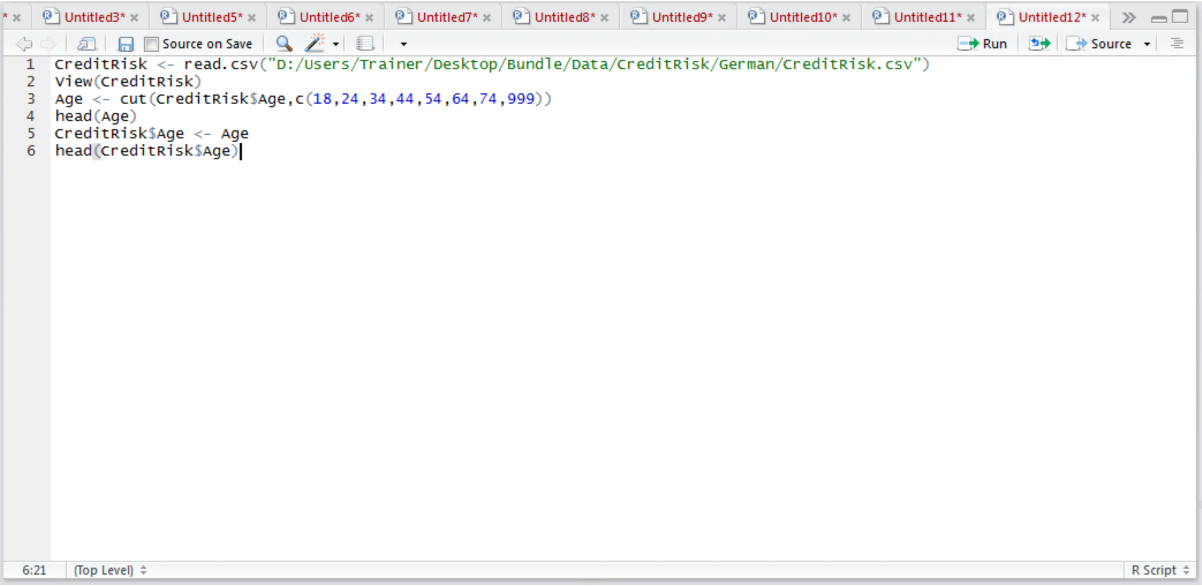


Run the line of script to console:

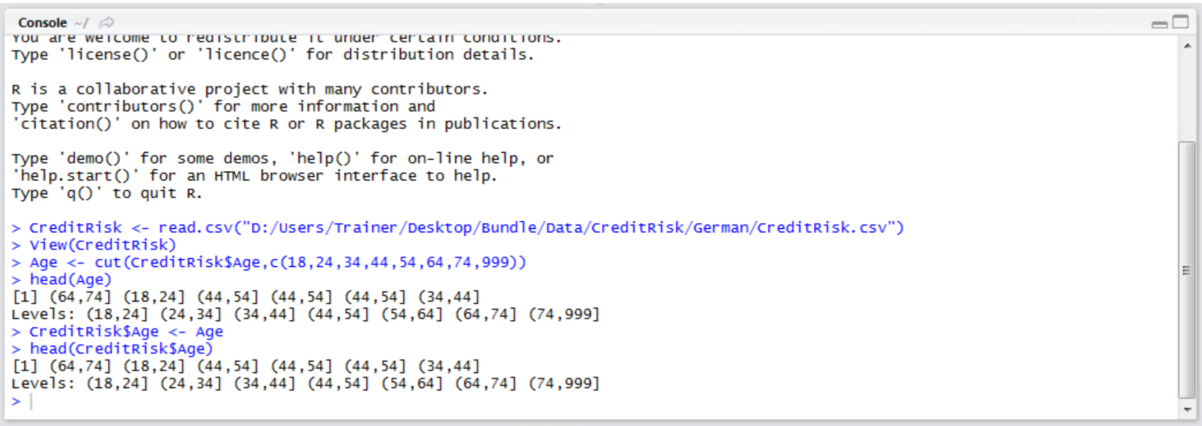


Check that the assignment has indeed transformed the CreditRisk$Age to a factor peeking the head() function:

head(CreditRisk$Age)



Run the line of script to console:



It can be seen that the continuous variable has been transformed.

Repeat for the remaining continuous variables, perhaps using the hist() function as described in procedure 55 to identify appropriate thresholds, as the following example:

#Bin

Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))

Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6,7,8,9,10,999))

Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))

Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))

Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))

Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))

#Allocate

CreditRisk$Requested_Amount <- Requested_Amount

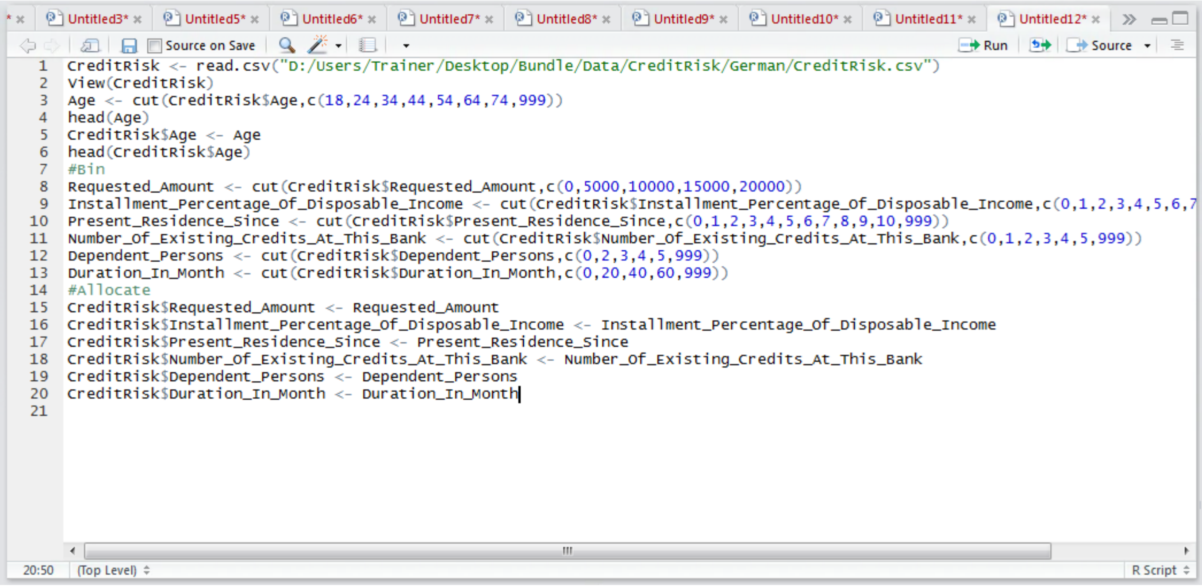
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income

CreditRisk$Present_Residence_Since <- Present_Residence_Since

CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank

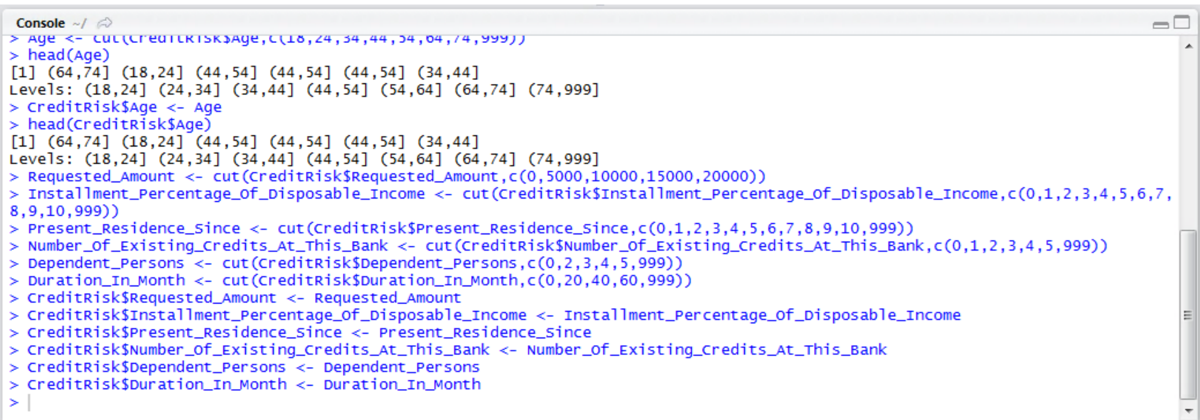
CreditRisk$Dependent_Persons <- Dependent_Persons

CreditRisk$Duration_In_Month <- Duration_In_Month



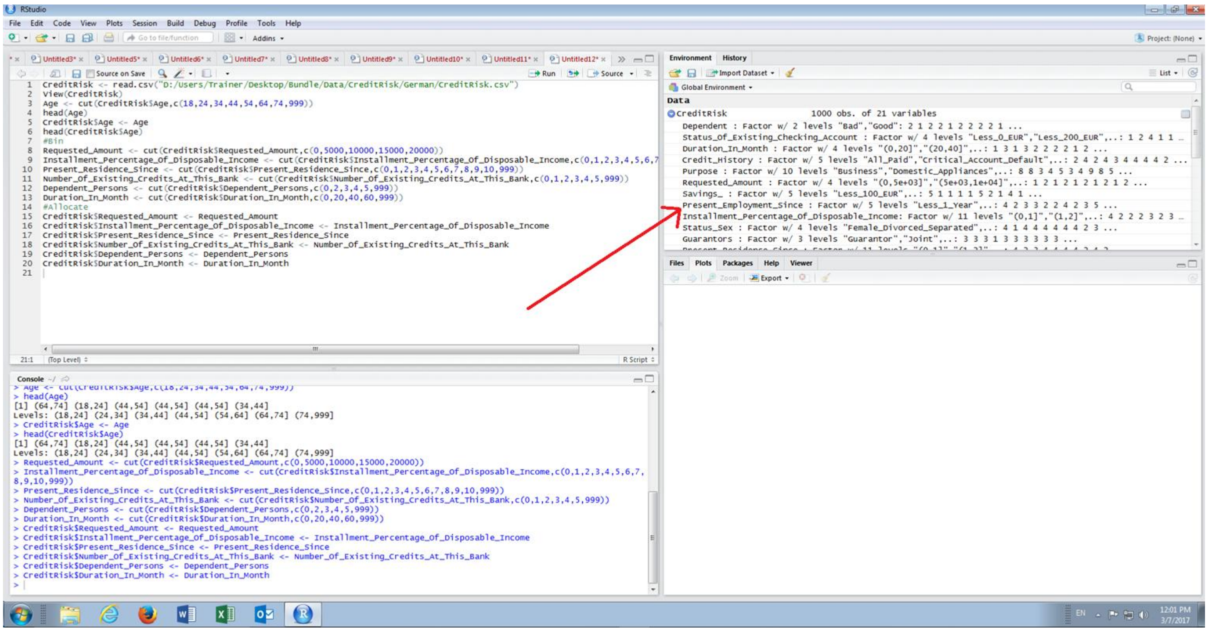
```
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
Age <- cut(CreditRisk$Age,c(18,24,34,44,54,64,74,999))
head(Age)
CreditRisk$Age <- Age
head(CreditRisk$Age)
#Bin
Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6,7
Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
#Allocate
CreditRisk$Requested_Amount <- Requested_Amount
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
CreditRisk$Present_Residence_Since <- Present_Residence_Since
CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
CreditRisk$Dependent_Persons <- Dependent_Persons
CreditRisk$Duration_In_Month <- Duration_In_Month
```

Run the block of script to console:



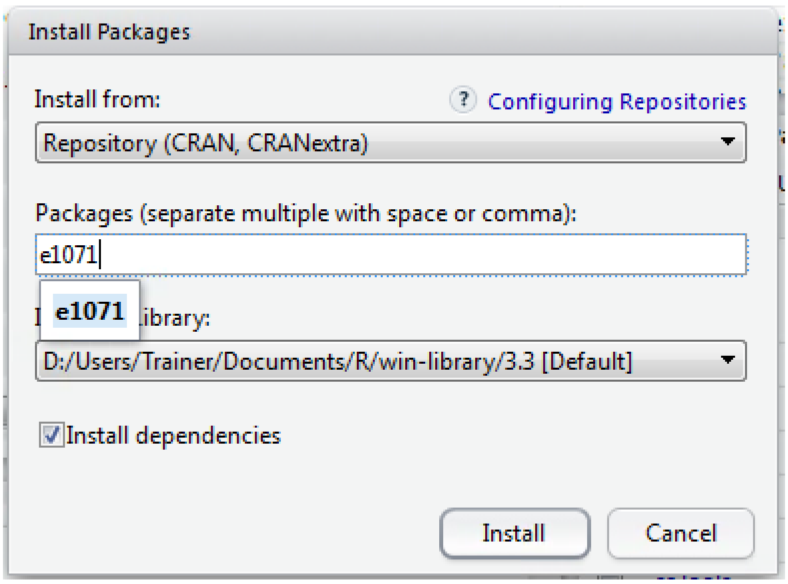
```
> head(Age)
[1] (64,74] (18,24] (44,54] (44,54] (44,54] (34,44]
Levels: (18,24] (24,34] (34,44] (44,54] (54,64] (64,74] (74,999]
> CreditRisk$Age <- Age
> head(CreditRisk$Age)
[1] (64,74] (18,24] (44,54] (44,54] (44,54] (34,44]
Levels: (18,24] (24,34] (34,44] (44,54] (54,64] (64,74] (74,999]
> Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
> Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6,7,
8,9,10,999))
> Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
> Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
> Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
> Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
> CreditRisk$Requested_Amount <- Requested_Amount
> CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
> CreditRisk$Present_Residence_Since <- Present_Residence_Since
> CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
> CreditRisk$Dependent_Persons <- Dependent_Persons
> CreditRisk$Duration_In_Month <- Duration_In_Month
>
```

It can be seen that from the data view pane in R studio, that for this data frame all components are now factors and so therefore appropriate for Bayesian Analysis:

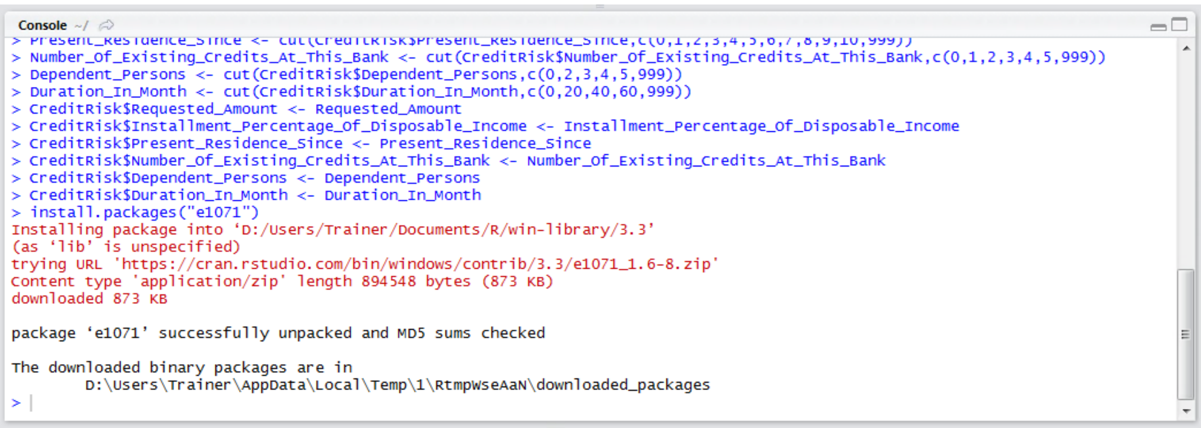


Procedure 2: Training a Naive Bayesian Classifier.

As a Naive Bayesian classifier is rather simple in its concept, all independent variables being treated and arcs flowing away from the dependent variable, it is to be expected that the process of training such a classifier is indeed trivial. To train a Bayesian model, simply pass the data frame, specify the factor that is to be treated as the dependent variable and the Laplace estimator (zero in this example). The naiveBayes() function exists as part of the e1071 package, a such begin by installing the package via RStudio:

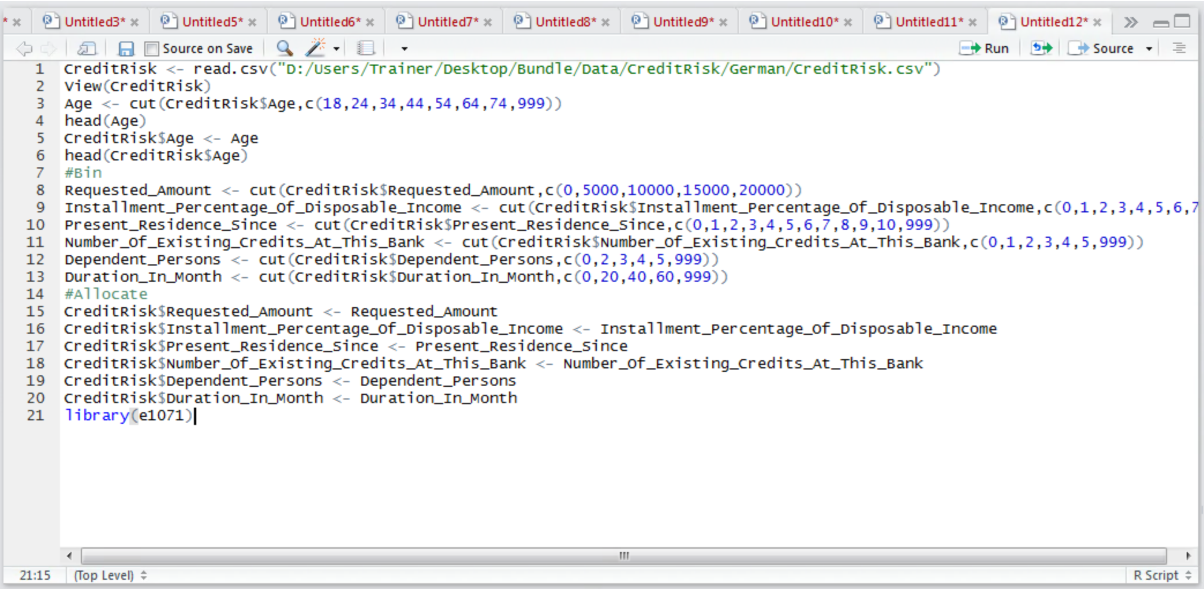


Click install to download and install this package:



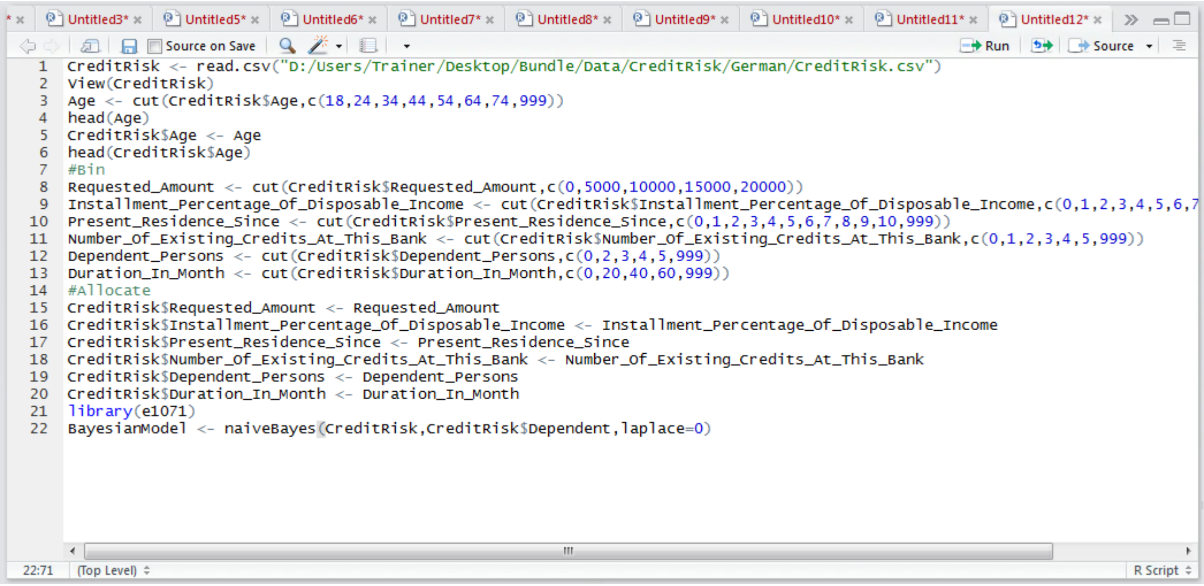
Reference the library:

library(e1071)



Run the line of script to console. To train a Naïve Bayesian model:

BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)



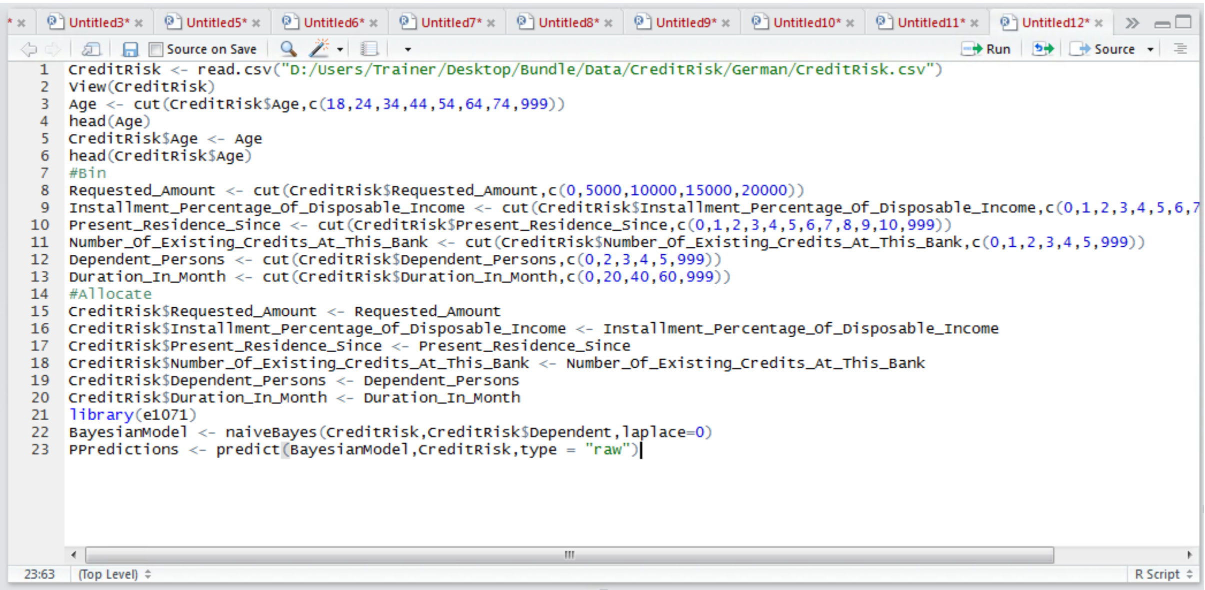
Run the line of script to console. The BayesModel object now contains a model that can be used to make P predictions as well as classifications.

Procedure 3: Recalling a Naive Bayesian Classifier for P.

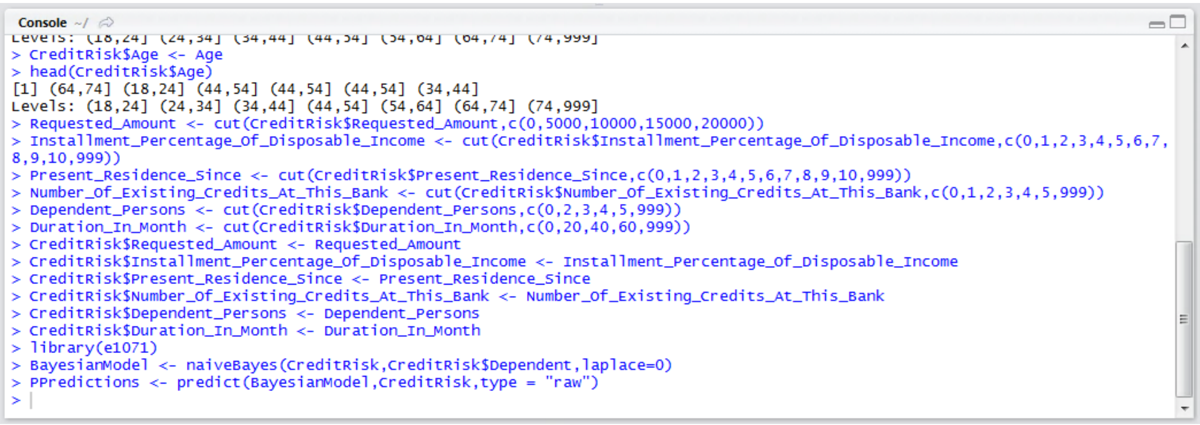
One of the benefits of using a Bayesian classifier is that it can return initiative probabilities which, ideally, should be fairly well calibrated to the actual environment. For example, suppose that a 30% P of rain is produced by a weather station for 100 days, if it were to rain on 30 of those days, that would be considered to be a well calibrated model. It follows that quite often it is not just the classification that is of interest, but the probability of a classification being accurate.

The familiar predict() function is available for use with the BayesModel object, the data frame to use in the recall and specifying a type to equal Raw, instructing the function to return P and not the most likely classification:

PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")



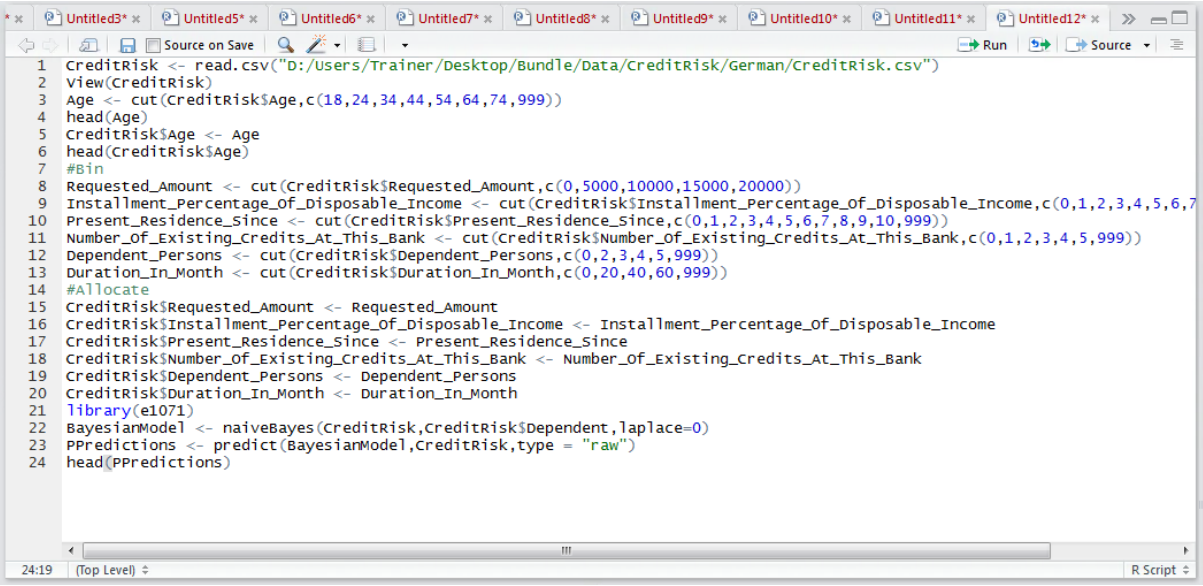
Run the line of script to console:



```
Levels: (18,24] (24,34] (34,44] (44,54] (54,64] (64,74] (74,999]
> CreditRisk$Age <- Age
> head(CreditRisk$Age)
[1] (64,74] (18,24] (44,54] (44,54] (44,54] (34,44]
Levels: (18,24] (24,34] (34,44] (44,54] (54,64] (64,74] (74,999]
> Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
> Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6,7,
8,9,10,999))
> Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
> Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
> Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
> Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
> CreditRisk$Requested_Amount <- Requested_Amount
> CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
> CreditRisk$Present_Residence_Since <- Present_Residence_Since
> CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
> CreditRisk$Dependent_Persons <- Dependent_Persons
> CreditRisk$Duration_In_Month <- Duration_In_Month
> library(e1071)
> BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
> PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
>
```

A peek of the data in the PPredictions output can be obtained via the head() function:

head(PPredictions)



```
CreditRisk <- read.csv("D:/Users/Trainer/Desktop/Bundle/Data/CreditRisk/German/CreditRisk.csv")
View(CreditRisk)
Age <- cut(CreditRisk$Age,c(18,24,34,44,54,64,74,999))
head(Age)
CreditRisk$Age <- Age
head(CreditRisk$Age)
#Bin
Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6,7
Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
#Allocate
CreditRisk$Requested_Amount <- Requested_Amount
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
CreditRisk$Present_Residence_Since <- Present_Residence_Since
CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
CreditRisk$Dependent_Persons <- Dependent_Persons
CreditRisk$Duration_In_Month <- Duration_In_Month
library(e1071)
BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
head(PPredictions)
```

Run the line of script to console:

```
> Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
> Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
> Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
> Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
> CreditRisk$Requested_Amount <- Requested_Amount
> CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
> CreditRisk$Present_Residence_Since <- Present_Residence_Since
> CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
> CreditRisk$Dependent_Persons <- Dependent_Persons
> CreditRisk$Duration_In_Month <- Duration_In_Month
> library(e1071)
> BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
> PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
> head(PPredictions)
              Bad         Good
[1,] 2.723594e-05 0.9999727641
[2,] 9.995426e-01 0.0004573721
[3,] 1.578426e-05 0.9999842157
[4,] 1.699783e-03 0.9983002170
[5,] 9.996594e-01 0.0003406426
[6,] 2.593379e-04 0.9997406621
>
```

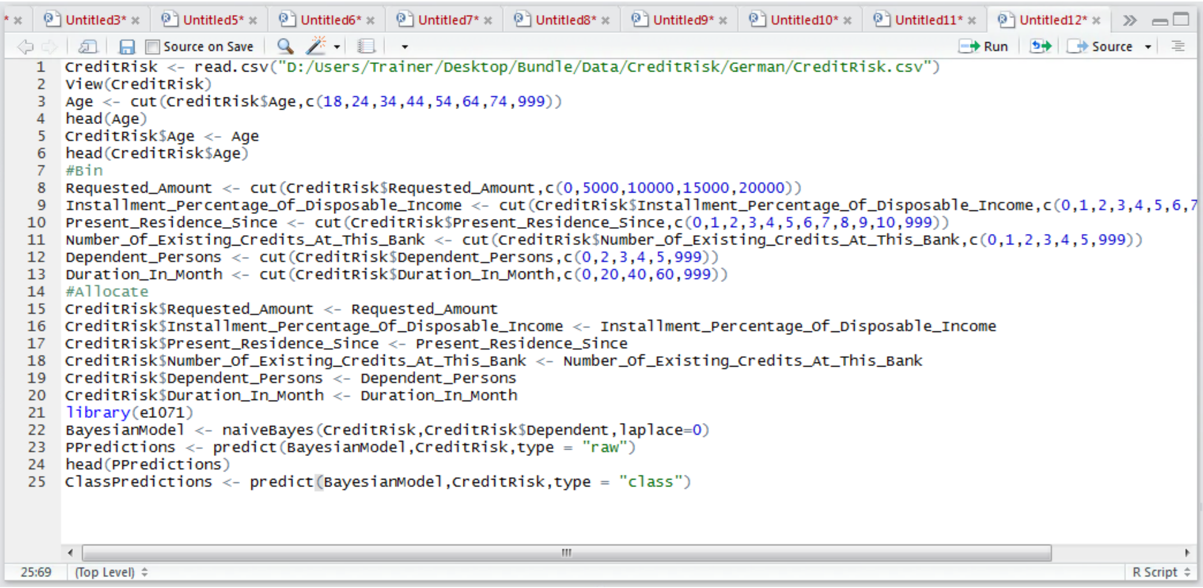
Horizontally the P will sum to one, and evidences clearly the most dominant class. Anecdotally, the calibration of P in naive Bayesian models can be somewhat disappointing, while the overarching classification and be surprisingly accurate.

JUBE

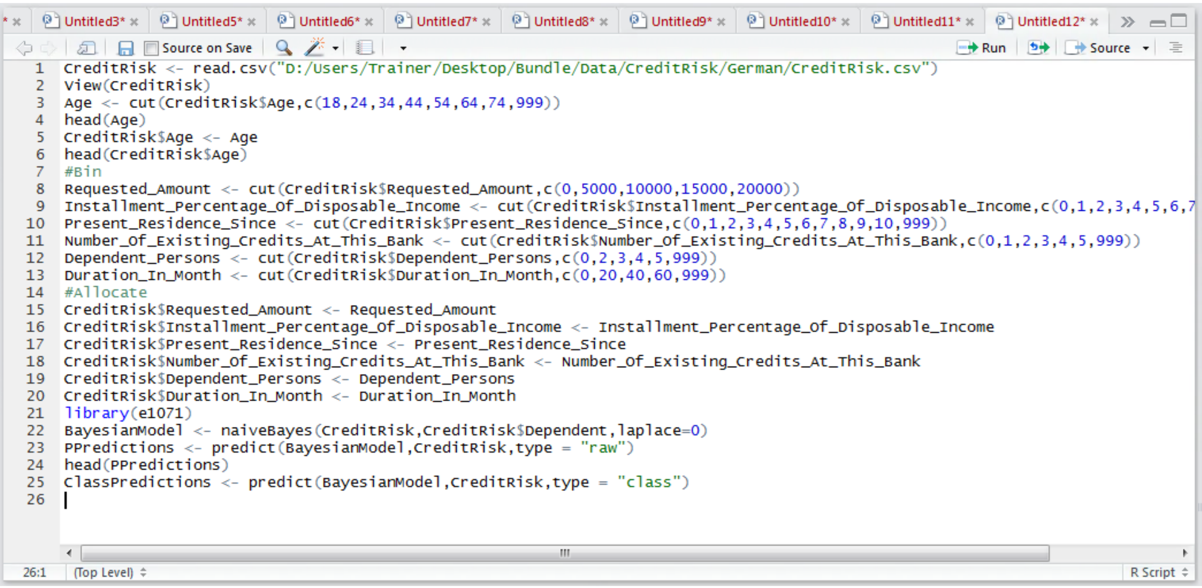
Procedure 4: Recalling a Naive Bayesian Classifier for Classification.

To recall the pivotal classification, rather than recall P for each class and drive it from the larger of the values, the type class can be specified:

ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")



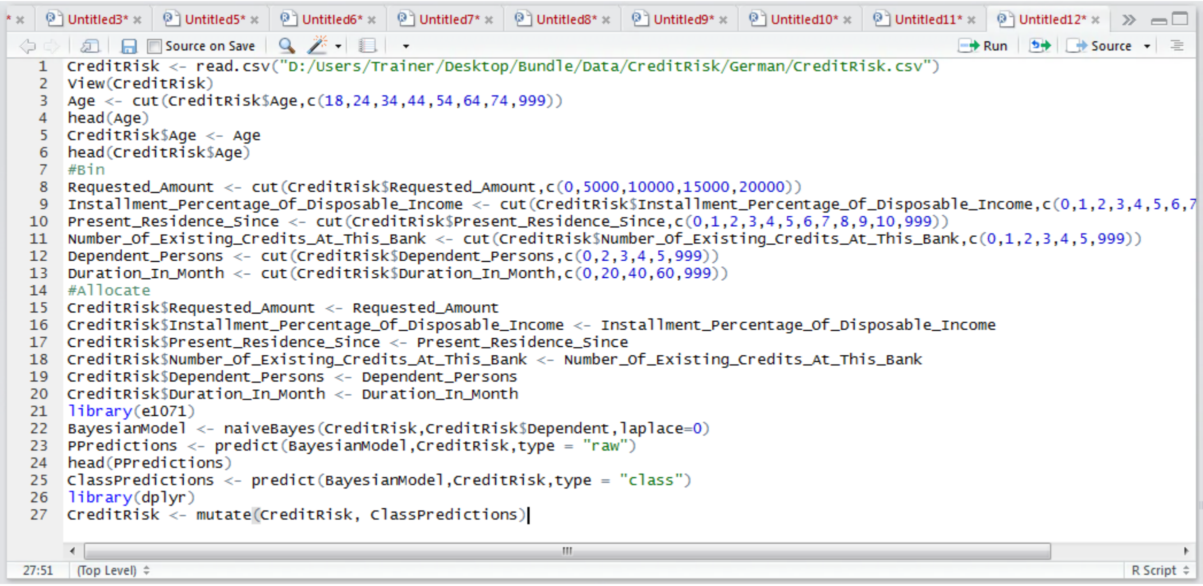
Run the line of script to console:



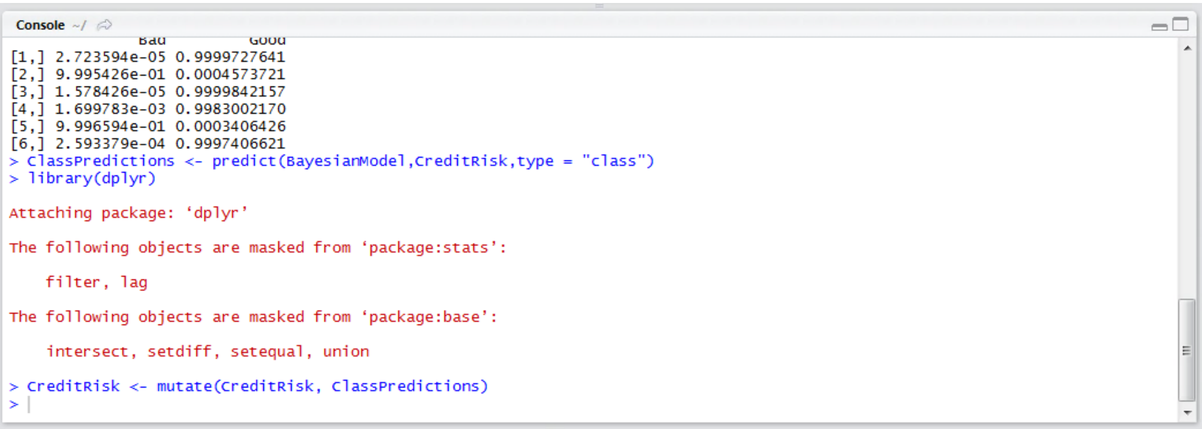
Merge the classification predictions into the CreditRisk data frame, specifying the dply library also:

library(dplyr)

CreditRisk <- mutate(CreditRisk, ClassPredictions)

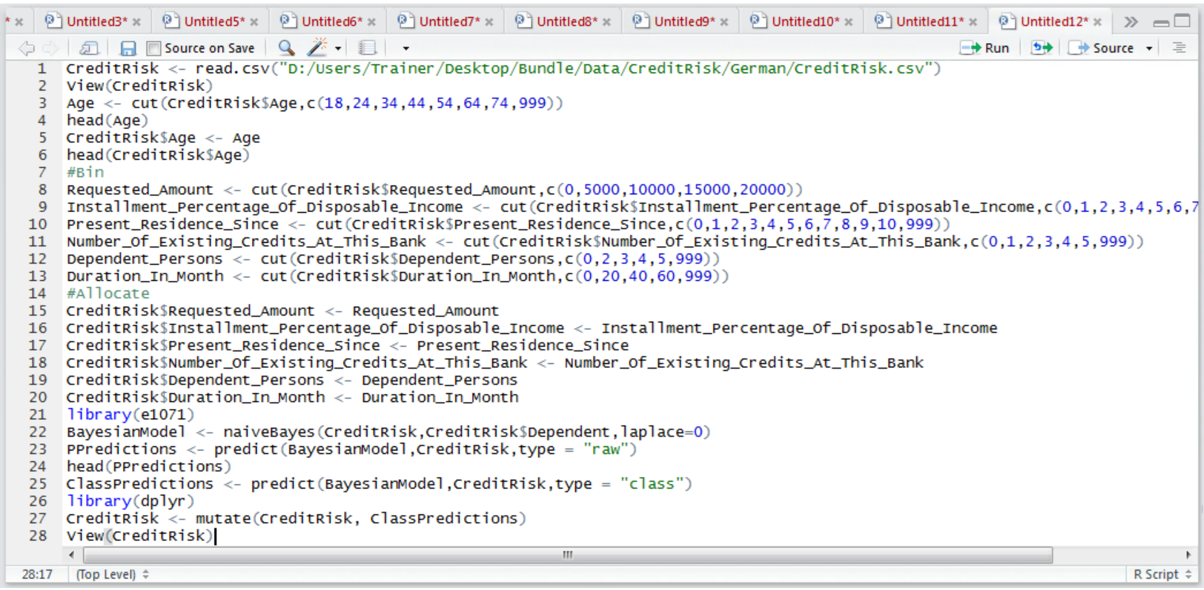


Run the line of script to console:



Viewing the CreditRisk data frame:

View(CreditRisk)



Run the line of script to console:

```
[1,] 2.723594e-05 0.9999727641
[2,] 9.995426e-01 0.0004573721
[3,] 1.578426e-05 0.9999842157
[4,] 1.699783e-03 0.9983002170
[5,] 9.996594e-01 0.0003406426
[6,] 2.593379e-04 0.9997406621
> ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> CreditRisk <- mutate(CreditRisk, ClassPredictions)
> View(CreditRisk)
>
```

Scroll to the last column in the RStudio viewer to reveal the classification for each record:

edit	Housing	Number_Of_Existing_Credits_At_This_Bank	Job	Dependent_Persons	Telephone	Foreign_Worker	ClassPredictions
	Owner	(1,2]	Skilled_Employee_Official	(0,2]	Yes_Own_Name	Yes	Good
	Owner	(0,1]	Skilled_Employee_Official	(0,2]	No	Yes	Bad
	Owner	(0,1]	Unskilled_Resident	(0,2]	No	Yes	Good
	Free	(0,1]	Skilled_Employee_Official	(0,2]	No	Yes	Good
	Free	(1,2]	Skilled_Employee_Official	(0,2]	No	Yes	Bad
	Free	(0,1]	Unskilled_Resident	(0,2]	Yes_Own_Name	Yes	Good
	Owner	[illegible]	Skilled_Employee_Official	(0,2]	No	Yes	Good
	Security	(0,1]	Management_Skilled	(0,2]	Yes_Own_Name	Yes	Good
	Owner	(0,1]	Unskilled_Resident	(0,2]	No	Yes	Good
	Owner	(1,2]	Management_Skilled	(0,2]	No	Yes	Bad
	Security	(0,1]	Skilled_Employee_Official	(0,2]	No	Yes	Bad
	Security	(0,1]	Skilled_Employee_Official	(0,2]	No	Yes	Bad
	Owner	(0,1]	Skilled_Employee_Official	(0,2]	Yes_Own_Name	Yes	Good
	Owner	(1,2]	Unskilled_Resident	(0,2]	No	Yes	Bad
	Security	(0,1]	Skilled_Employee_Official	(0,2]	No	Yes	Good
	Owner	(0,1]	Unskilled_Resident	(0,2]	No	Yes	Bad
	Owner	(1,2]	Skilled_Employee_Official	(0,2]	No	Yes	Good

Showing 1 to 18 of 1,000 entries

Procedure 5: Create a Naive Bayesian Network with a Laplace Estimator.

To create a Bayesian model with a nominal Laplace estimator of 1, which will mean that in the event that there is nothing it is switch to at least one occurrence in the observation, simply change the parameter value in the training:

SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)

```
View(CreditRisk)
Age <- cut(CreditRisk$Age,c(18,24,34,44,54,64,74,999))
head(Age)
CreditRisk$Age <- Age
head(CreditRisk$Age)
#Bin
Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,
Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
#Allocate
CreditRisk$Requested_Amount <- Requested_Amount
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
CreditRisk$Present_Residence_Since <- Present_Residence_Since
CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
CreditRisk$Dependent_Persons <- Dependent_Persons
CreditRisk$Duration_In_Month <- Duration_In_Month
library(e1071)
BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
head(PPredictions)
ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
library(dplyr)
CreditRisk <- mutate(CreditRisk, ClassPredictions)
View(CreditRisk)
SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)
```

Run the line of script to console:

```
[3,] 1.578426e-05 0.9999842157
[4,] 1.699783e-03 0.9983002170
[5,] 9.996594e-01 0.0003406426
[6,] 2.593379e-04 0.9997406621
> ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> CreditRisk <- mutate(CreditRisk, ClassPredictions)
> View(CreditRisk)
> SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)
>
```

A Bayesian model has been created as SafeBaysianModel. Recall the model:

ClassPredictions <- predict(SafeBayesianModel,CreditRisk,type = "class")

```
head(Age)
CreditRisk$Age <- Age
head(CreditRisk$Age)
#Bin
Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,
Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
#Allocate
CreditRisk$Requested_Amount <- Requested_Amount
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
CreditRisk$Present_Residence_Since <- Present_Residence_Since
CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
CreditRisk$Dependent_Persons <- Dependent_Persons
CreditRisk$Duration_In_Month <- Duration_In_Month
library(e1071)
BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
head(PPredictions)
ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
library(dplyr)
CreditRisk <- mutate(CreditRisk, ClassPredictions)
View(CreditRisk)
SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)
ClassPredictions <- predict(SafeBayesianModel,CreditRisk,type = "class")
```

Run the line of script to console:

```
Console ~/
[4,] 1.699783e-03 0.9983002170
[5,] 9.996594e-01 0.0003406426
[6,] 2.593379e-04 0.9997406621
> ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> CreditRisk <- mutate(CreditRisk, ClassPredictions)
> View(CreditRisk)
> SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)
> ClassPredictions <- predict(SafeBayesianModel,CreditRisk,type = "class")
> |
```

The de-facto method to appraise the performance of the model would be to create a confusion matrix as procedure 100:

library(gmodels)

CrossTable(CreditRisk$Dependent, ClassPredictions)

```
CreditRisk$Age <- Age
head(CreditRisk$Age)
#Bin
Requested_Amount <- cut(CreditRisk$Requested_Amount,c(0,5000,10000,15000,20000))
Installment_Percentage_Of_Disposable_Income <- cut(CreditRisk$Installment_Percentage_Of_Disposable_Income,c(0,1,2,3,4,5,6
Present_Residence_Since <- cut(CreditRisk$Present_Residence_Since,c(0,1,2,3,4,5,6,7,8,9,10,999))
Number_Of_Existing_Credits_At_This_Bank <- cut(CreditRisk$Number_Of_Existing_Credits_At_This_Bank,c(0,1,2,3,4,5,999))
Dependent_Persons <- cut(CreditRisk$Dependent_Persons,c(0,2,3,4,5,999))
Duration_In_Month <- cut(CreditRisk$Duration_In_Month,c(0,20,40,60,999))
#Allocate
CreditRisk$Requested_Amount <- Requested_Amount
CreditRisk$Installment_Percentage_Of_Disposable_Income <- Installment_Percentage_Of_Disposable_Income
CreditRisk$Present_Residence_Since <- Present_Residence_Since
CreditRisk$Number_Of_Existing_Credits_At_This_Bank <- Number_Of_Existing_Credits_At_This_Bank
CreditRisk$Dependent_Persons <- Dependent_Persons
CreditRisk$Duration_In_Month <- Duration_In_Month
library(e1071)
BayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=0)
PPredictions <- predict(BayesianModel,CreditRisk,type = "raw")
head(PPredictions)
ClassPredictions <- predict(BayesianModel,CreditRisk,type = "class")
library(dplyr)
CreditRisk <- mutate(CreditRisk, ClassPredictions)
View(CreditRisk)
SafeBayesianModel <- naiveBayes(CreditRisk,CreditRisk$Dependent,laplace=1)
ClassPredictions <- predict(SafeBayesianModel,CreditRisk,type = "class")
library(gmodels)
CrossTable(CreditRisk$Dependent, ClassPredictions)
```

Run the block of script to console:

```
Console ~/
                     | ClassPredictions 
CreditRisk$Dependent |       Bad |      Good | Row Total | 
---------------------|-----------|-----------|-----------|
                 Bad |       300 |         0 |       300 | 
                     |   490.000 |   210.000 |           | 
                     |     1.000 |     0.000 |     0.300 | 
                     |     1.000 |     0.000 |           | 
                     |     0.300 |     0.000 |           | 
---------------------|-----------|-----------|-----------|
                Good |         0 |       700 |       700 | 
                     |   210.000 |    90.000 |           | 
                     |     0.000 |     1.000 |     0.700 | 
                     |     0.000 |     1.000 |           | 
                     |     0.000 |     0.700 |           | 
---------------------|-----------|-----------|-----------|
        Column Total |       300 |       700 |      1000 | 
                     |     0.300 |     0.700 |           | 
---------------------|-----------|-----------|-----------|

> |
```

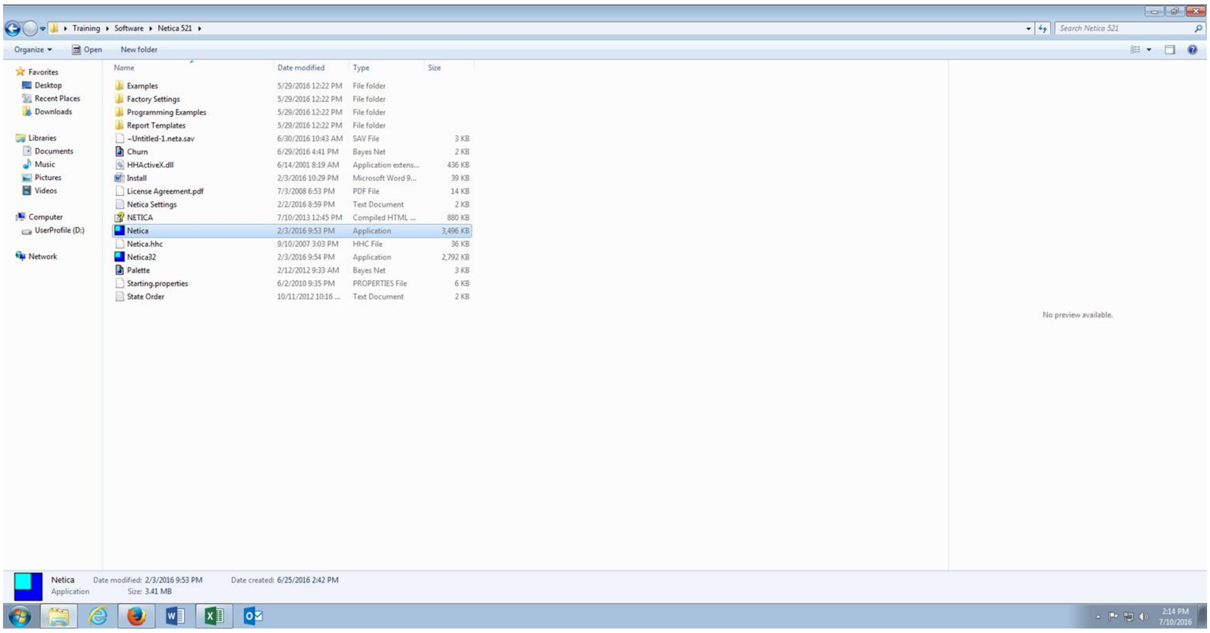
It can be seen that this naive Bayesian model appears to be startlingly accurate, which stands to reason as the same data is being used to test as was trained. It follows that this would benefit from an element of cross validation, which was introduced in procedure 113 when Gradient Boosting Machines were visited.

Module 12: Norsys Netica and Bayesian Analysis.

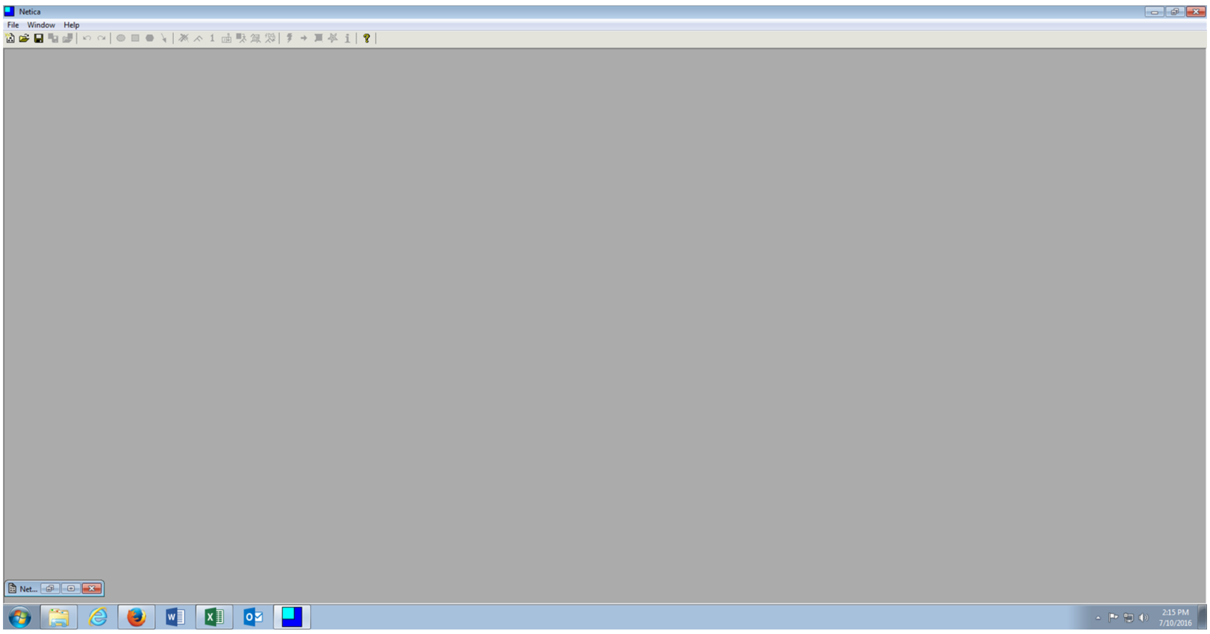
Norsys Netica is a modelling tool that allows for the creative development of Bayesian networks based on either belief (this would be subjective probability) or data (taking a frequentist approach to probability as available in data).

The software does not install nativity to the operating system, the executables are in the directory:

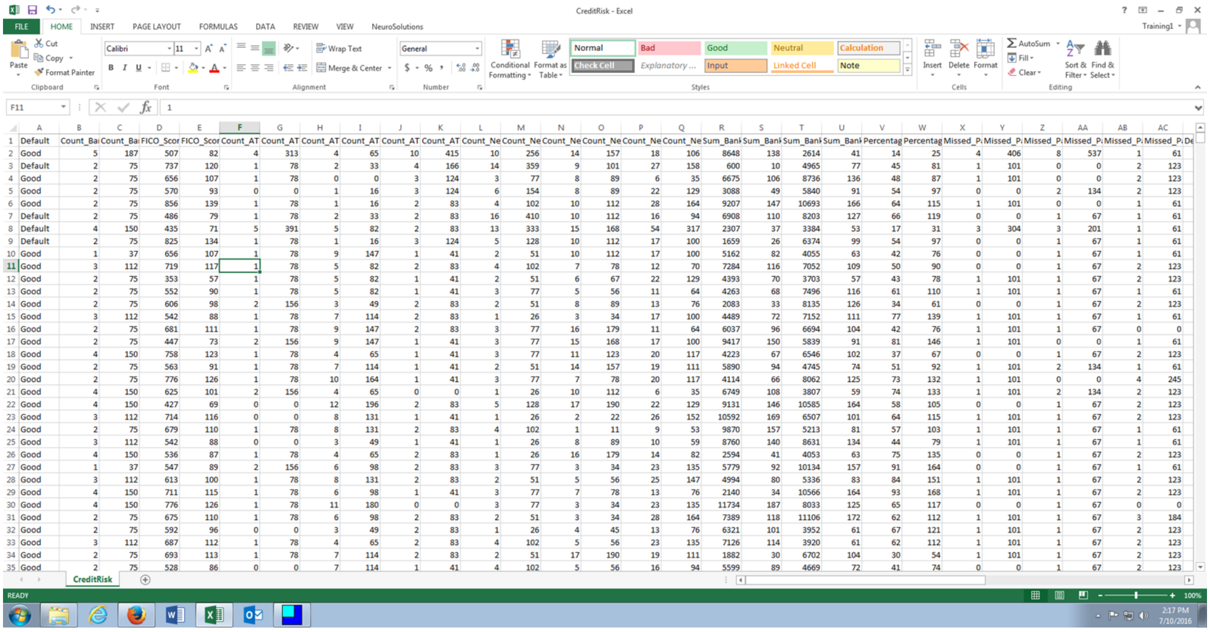
\Training\Software\Netica 521



Execute the program Netica.exe, which will open the Netica user interface:



The data file that will be used in these procedures is available in Training\Data\CreditRisk and is titled CreditRisk.csv:



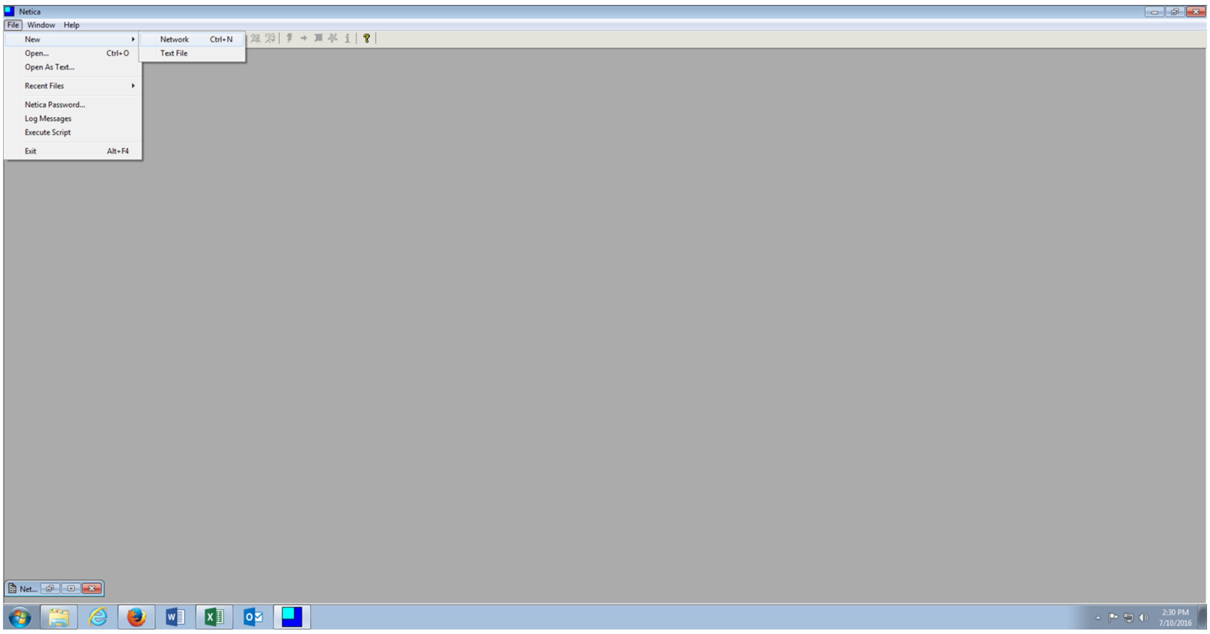
The CreditRisk.csv file is extremely large containing an uneven number of default vs. good cases, it could be said that this is a representative sample unlike the logistic regression techniques with rely on an even number of cases in both dispositions.

IN DESIGN TIME NETICA OFTEN HAD BUGS AND CAN CRASH. BE SURE TO SAVE WORK REGUALLY.

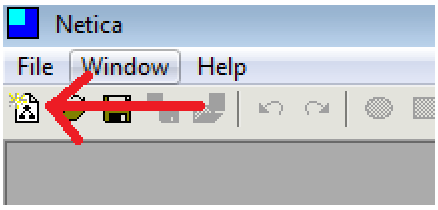
Procedure 1: Create a New Canvas, add a Dependent Variable and an Independent Variable.

Like Decision Trees, Netica is quite visual. Independent and Dependent variables are stamped to a canvas and joined together in the direction of causation, creating a network. The starting point for creating a Bayesian Network is to create a new canvas.

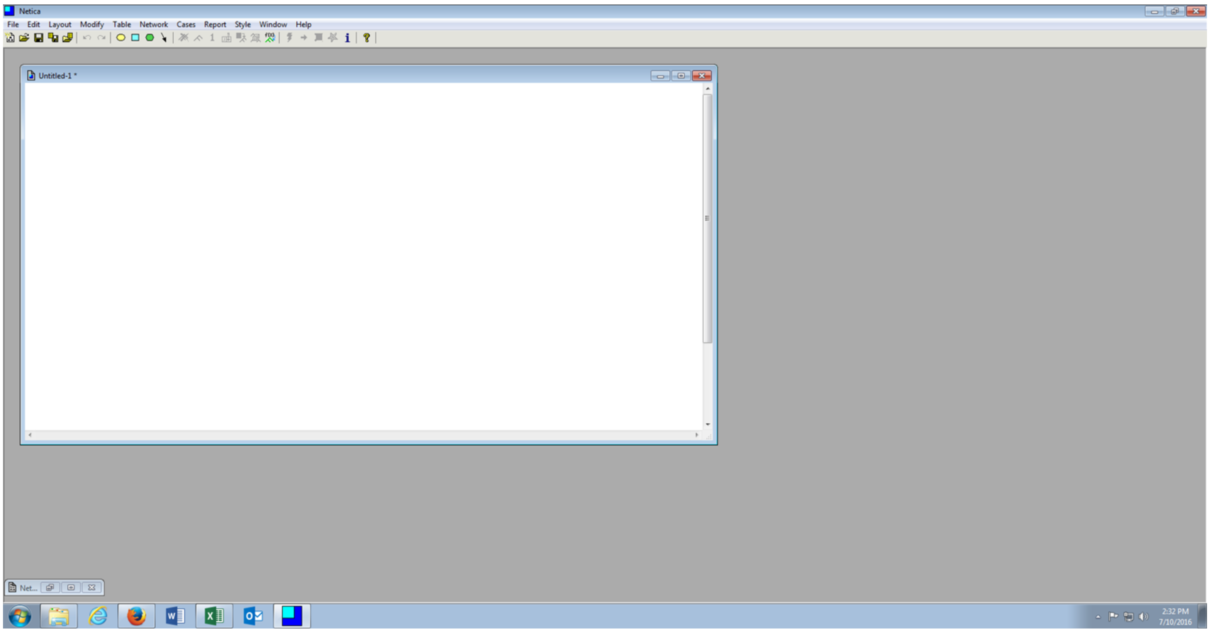
Creating a new canvas is achieved from the File menu, by clicking File….New….Network:



Creating a new canvas can also be achieved by clicking the icon as follows:

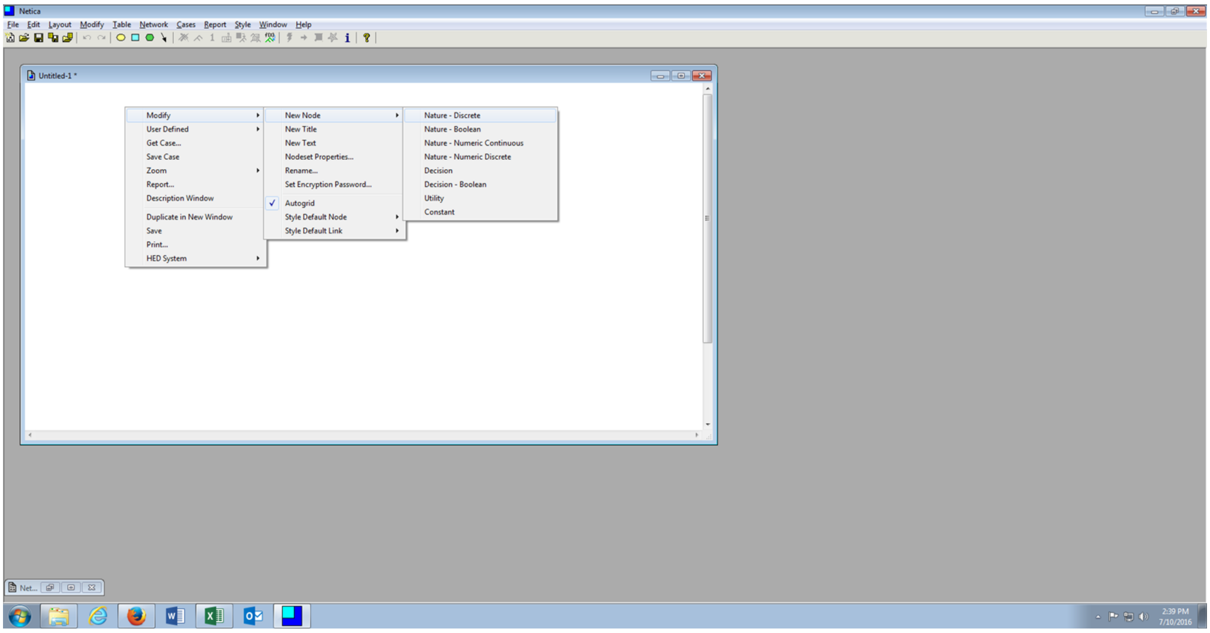


A new canvas will appear:

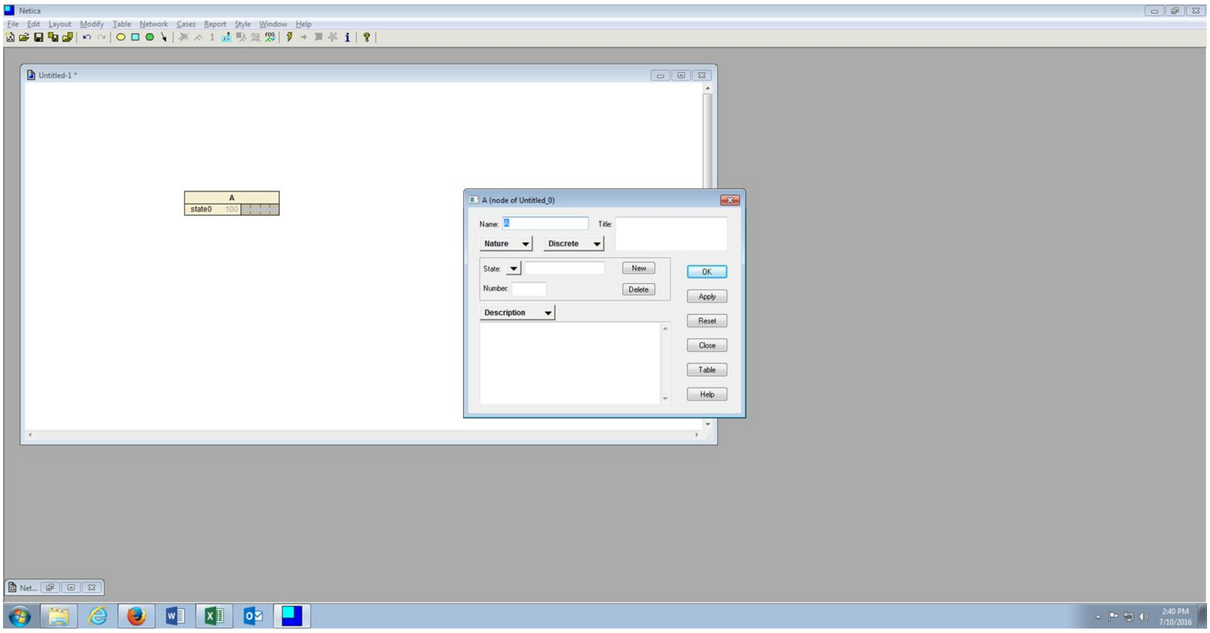


Variables, hitherto nodes, are stamped to the canvas with one node for each variable to be included in the model. In this example there will be a single node representing the dependent variable and a single node representing the independent variable.

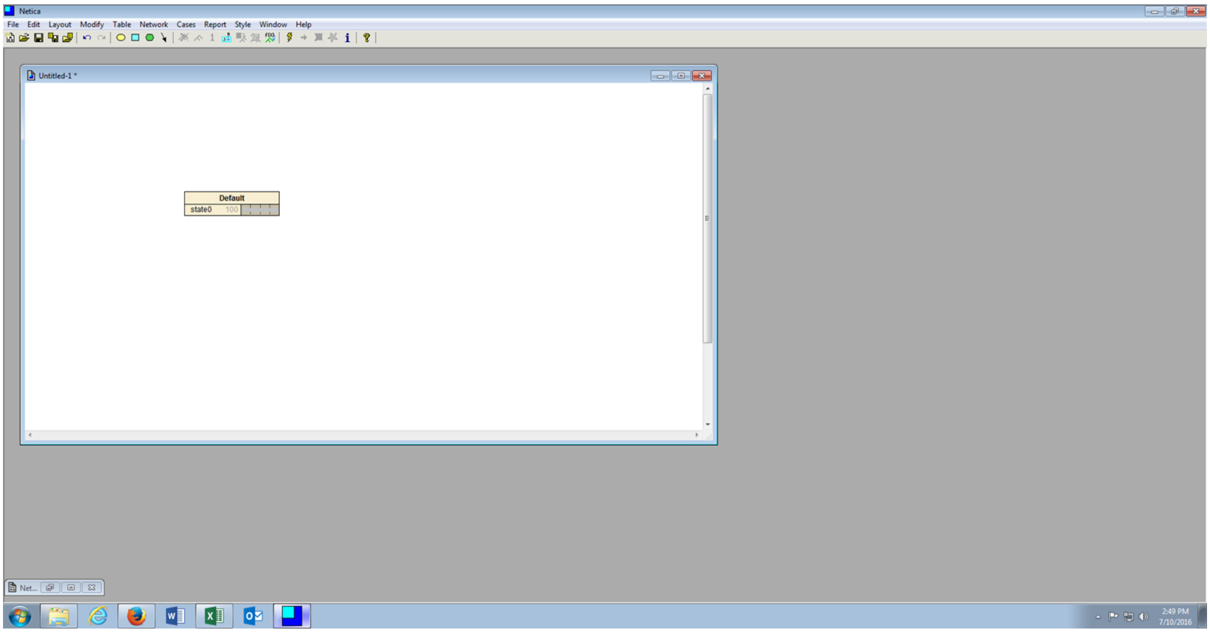
Right click on the canvas and expand the Modify Menu by right clicking, then clicking New Node, then clicking Nature Node Discrete:



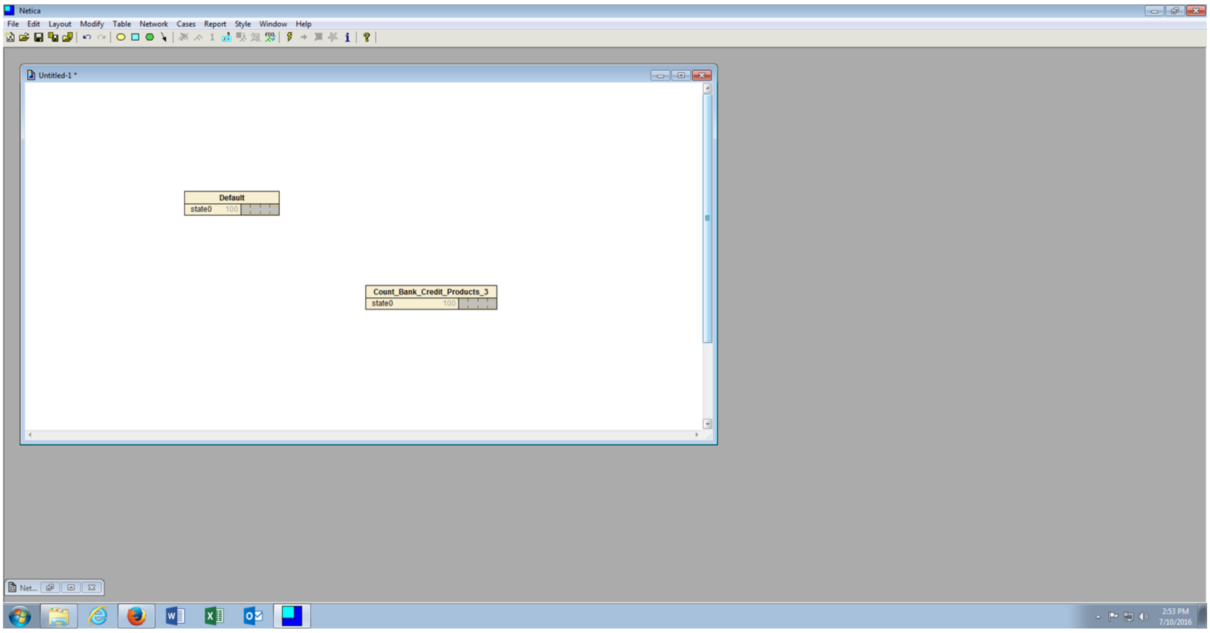
A node will be stamped to the canvas in the location of the right click with the nodes properties box being shown by default:



Name the variable to EXACTLY the same as the dependent variable is named in the dataset, in this example Default, then click OK:



Repeat the process adding a second node to the canvas, this time naming the node as an independent variable with yes \ no states, in this example Count_Bank_Credit_Products_Greater_3:

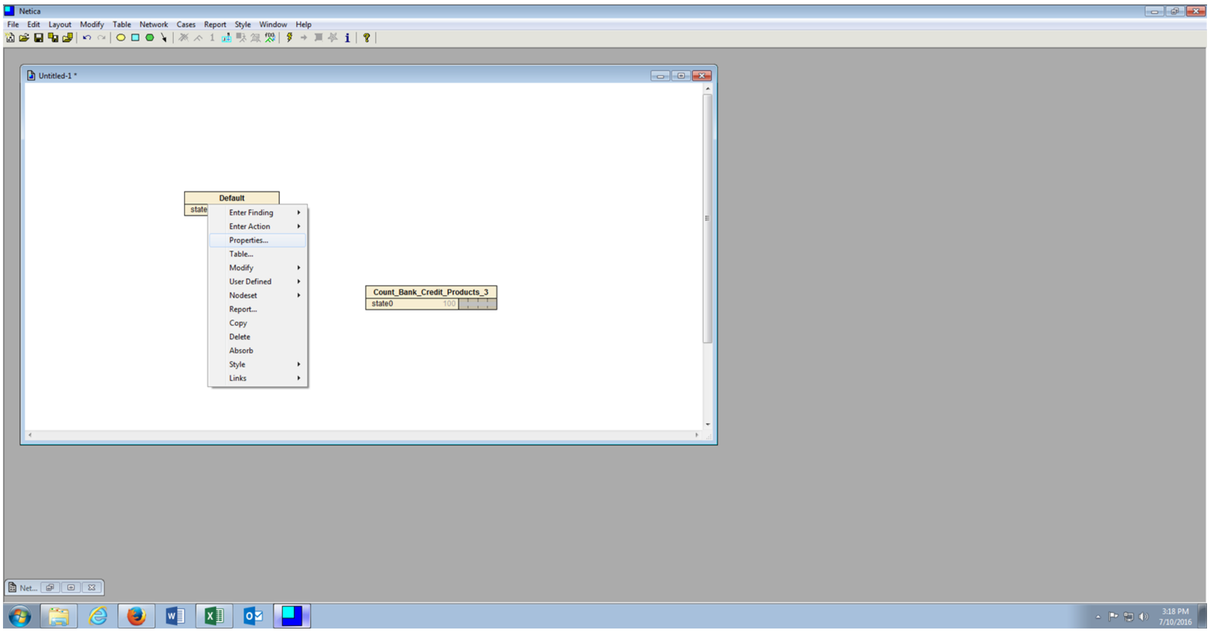


Notice that there are now two nodes stamped to the canvas, one for the dependent variable and one for the independent variable. Notice also that while each of these nodes has two possible values, referred to as states, the nodes only reflect one default state.

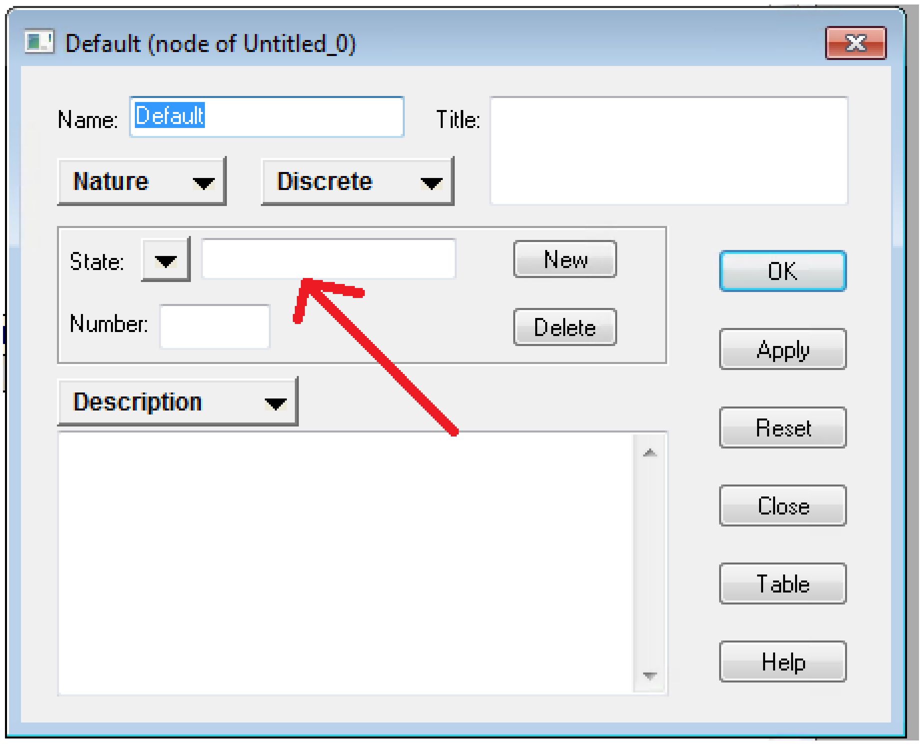
Procedure 2: Set States attributed to the Dependent and Independent Variables.

For both of the nodes stamped to the canvas, representing a single dependent variable or a single independent variable, there is the same states of Yes \ No (i.e. both nodes only have two possible, string based outcomes). It follows that each of the nodes needs to have the Yes \ No states set.

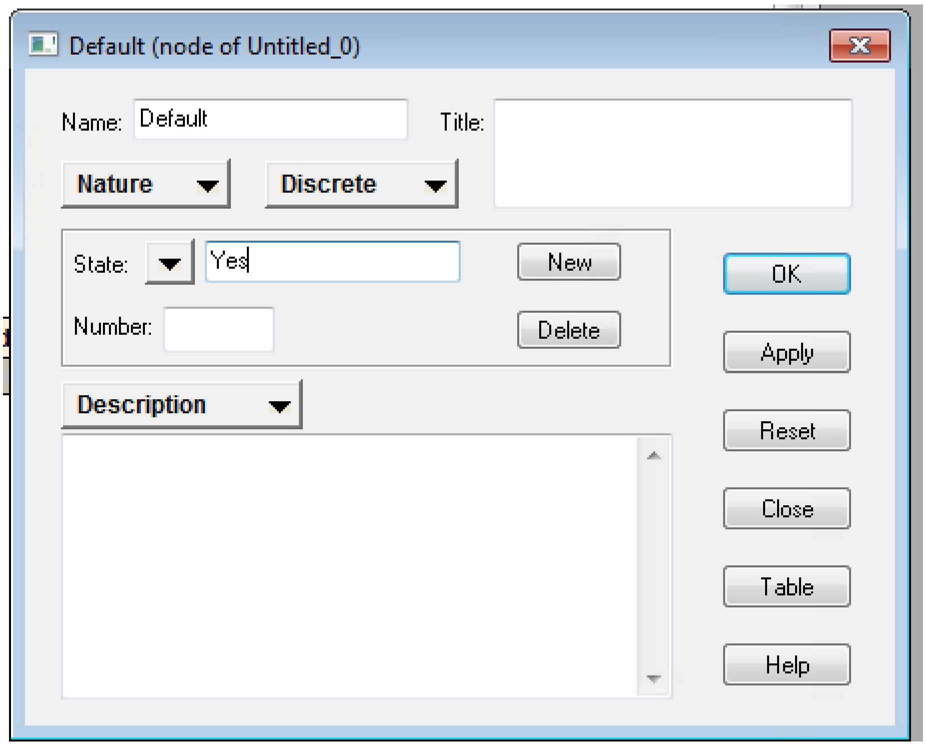
To set the states of a node, right click on the node and select properties, in this case right click on the Default node (the dependent variable):



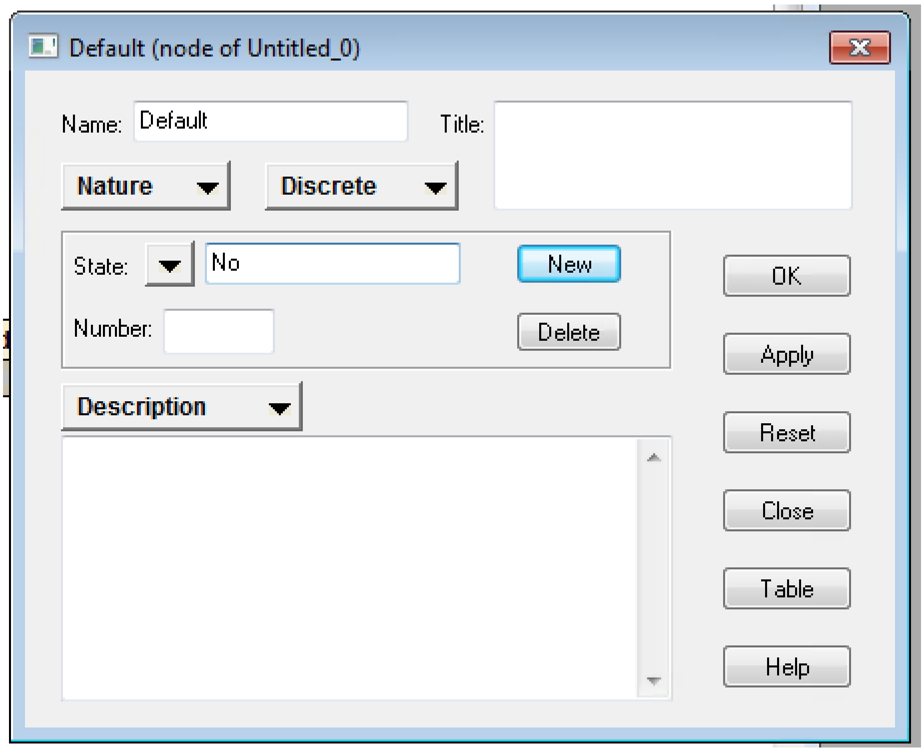
The properties window will open which is the same windows used to name the node. Focussing attention towards the centre of the window, there is an entry box titled State:



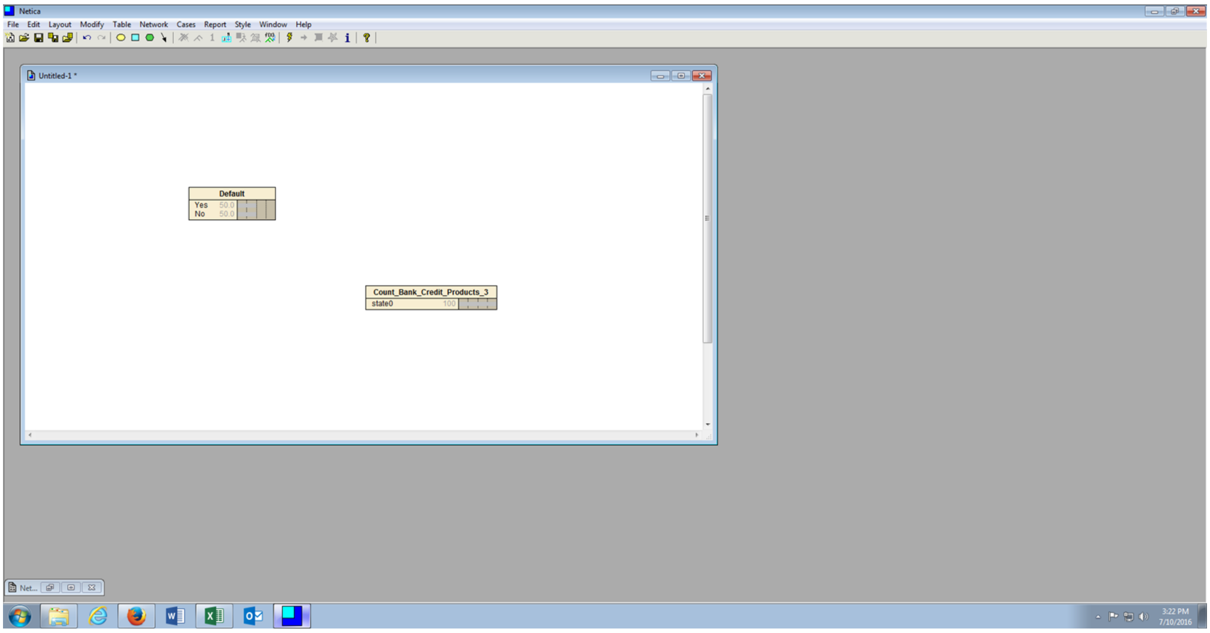
Type the name of the first state, which would be Yes:



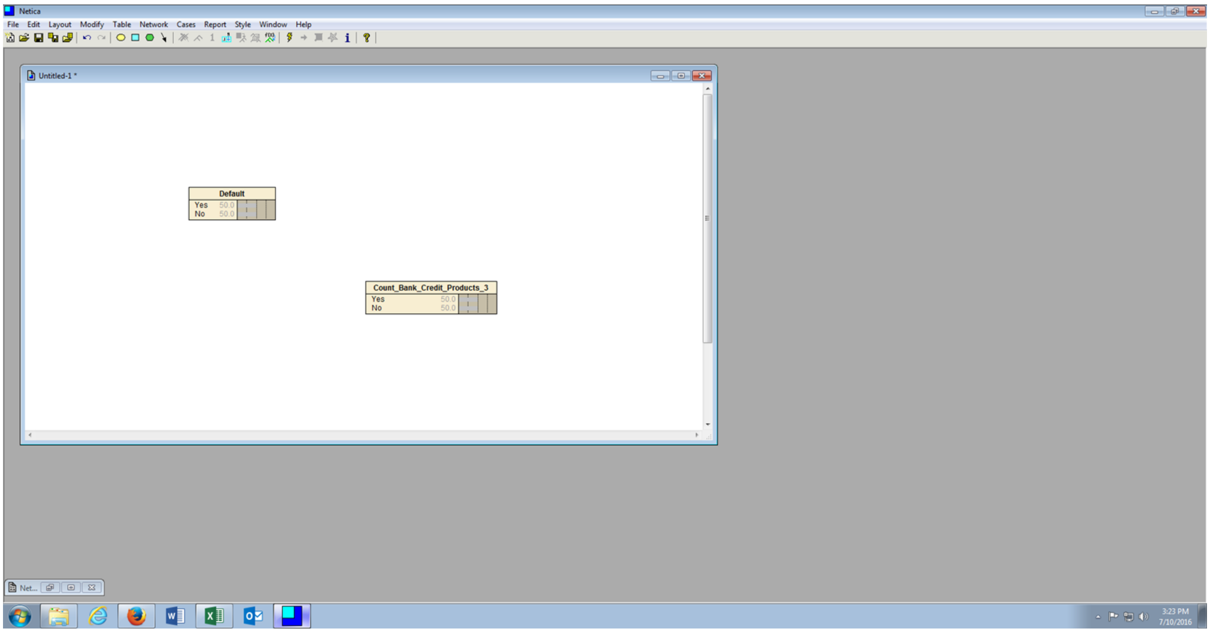
Then click New to commit the Yes state, proceeding to create the No state:



Click OK to commit both states to the node, after which the Node will be updated to reflect both states with an even probability:



Repeat the process for each node on the canvas, for each possible state for that node:

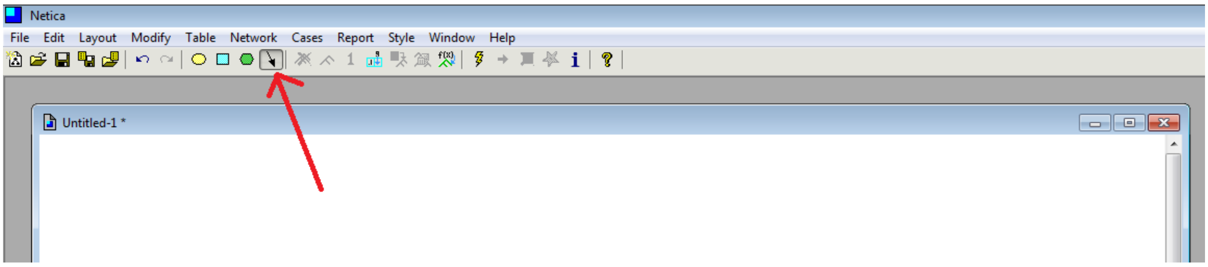


Procedure 3: Link Variables as causes consequence.

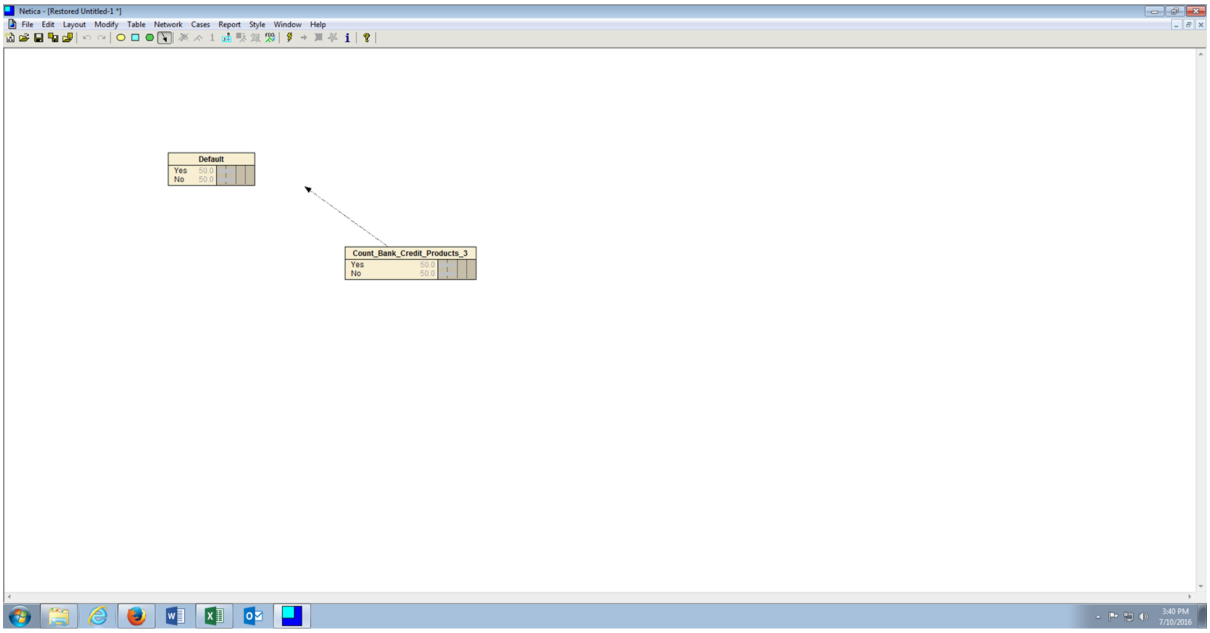
One method of creating Bayesian Networks is to judge an Independent Variable to cause a consequence to another, most likely dependent, variable.

To reflect that one variable can cause a consequence for another variable, in this example Count_Bank_Credit_Products_3 having consequence for Default, a link is drawn between the variables. Links always flow in the direction of causation.

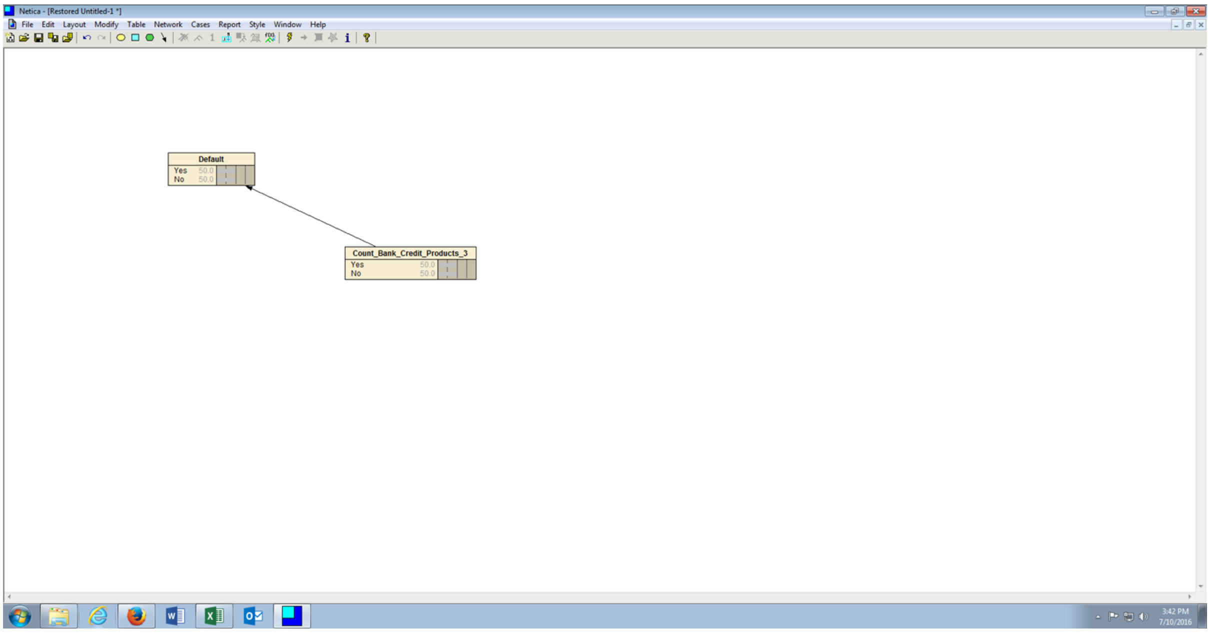
To add a link, click the link button on the icon menu:



After the link icon is toggled, click in the centre of the node that is causing a consequence, in this case Count_Bank_Credit_Products_3 then drag:



Drag the link to the centre of the node which suffers consequence, in this case Default, then drop to consummate the link:



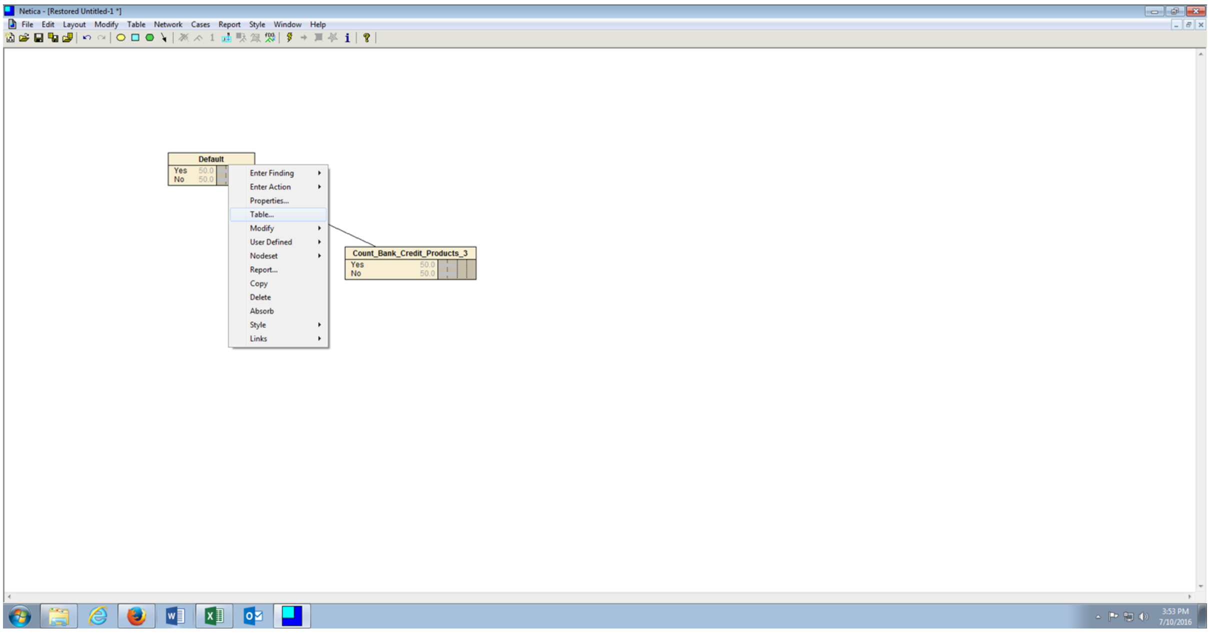
Following the causes consequence paradigm makes the construction of node probability tables more intuitive (as the tables will be built at the consequence inferring all possible scenarios). Repeat the links for every node that causes a default consequence on the canvas.

This approach constructs what is known as a naive Bayesian Network, in that all nodes evenly cause a single consequence in structure.

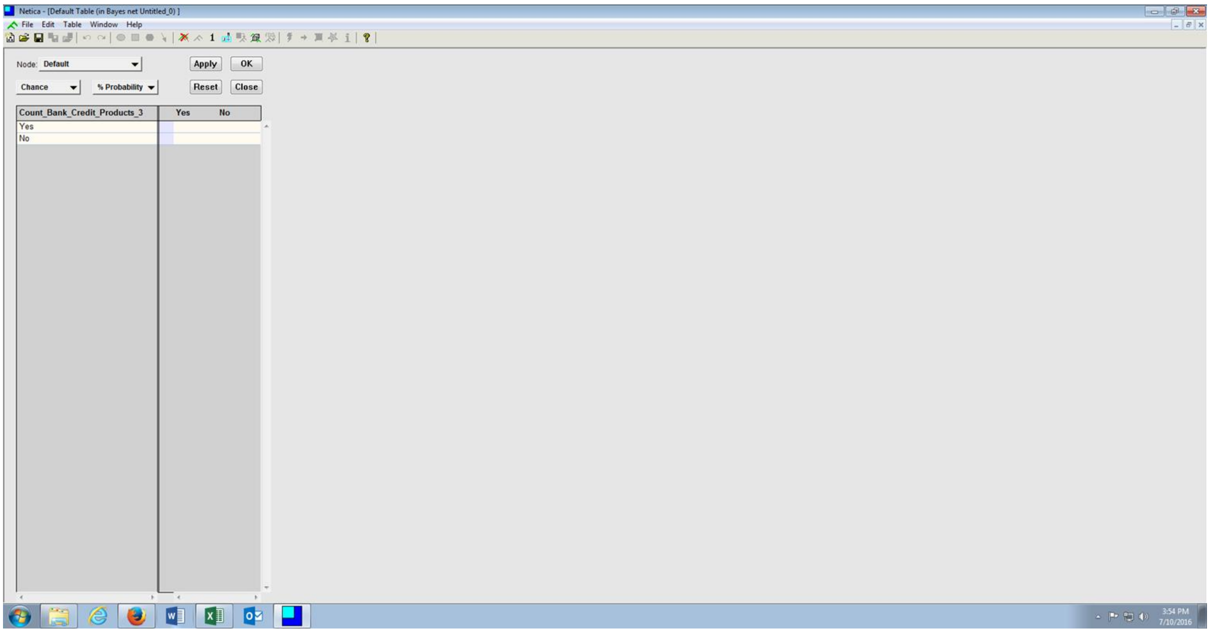
Procedure 4: Enter subjective probabilities for each consequence.

In creating a consequence, with many potential causes, and with the causes being state based (which in this example is Yes \ No), a finite set of scenarios that cause a consequence can now be inferred by Netica.

To view the finite scenarios that can cause a consequence, right click on the consequence node, in this case Default, the click Table (short for Node \ Conditional Probability Table):



The node probability infers every possible scenario in the Bayesian Network, calling for subjective probabilities to be included:

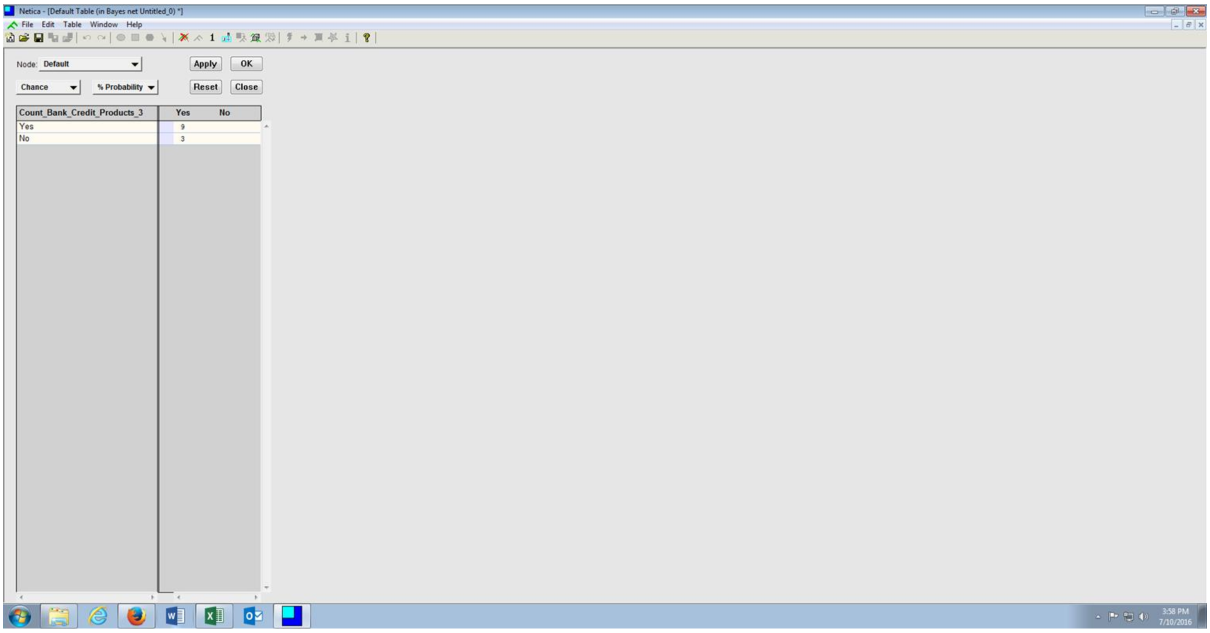


In this simple example, there are two scenarios which require subjective probabilities, however, with more nodes this GREATLY expands. Subjective probability needs to be apportioned to each scenario, rather belief (hence Bayesian Belief Networks).

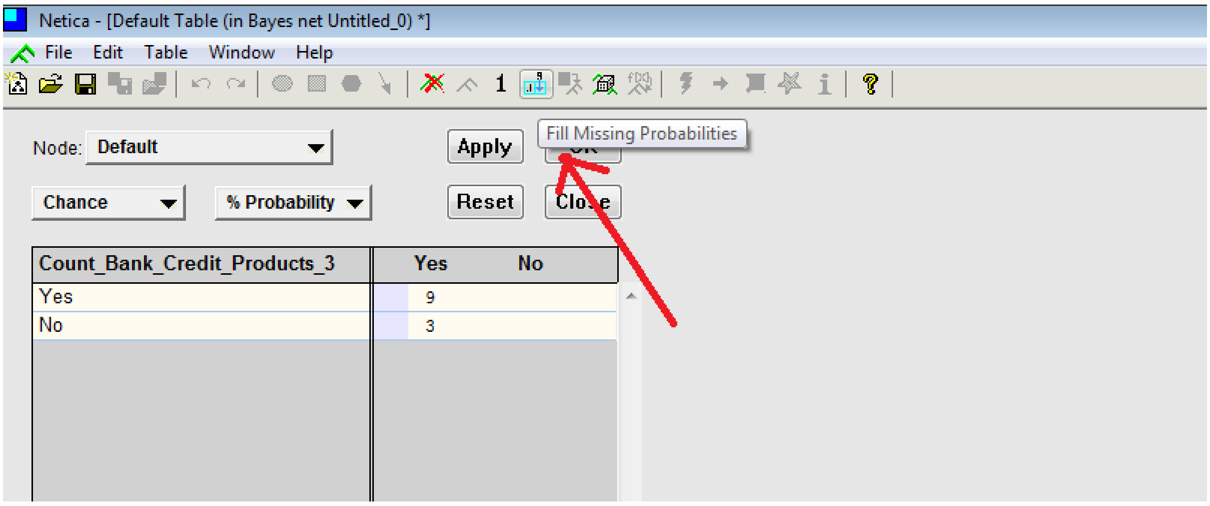
In this example apportion the following subjective probability:

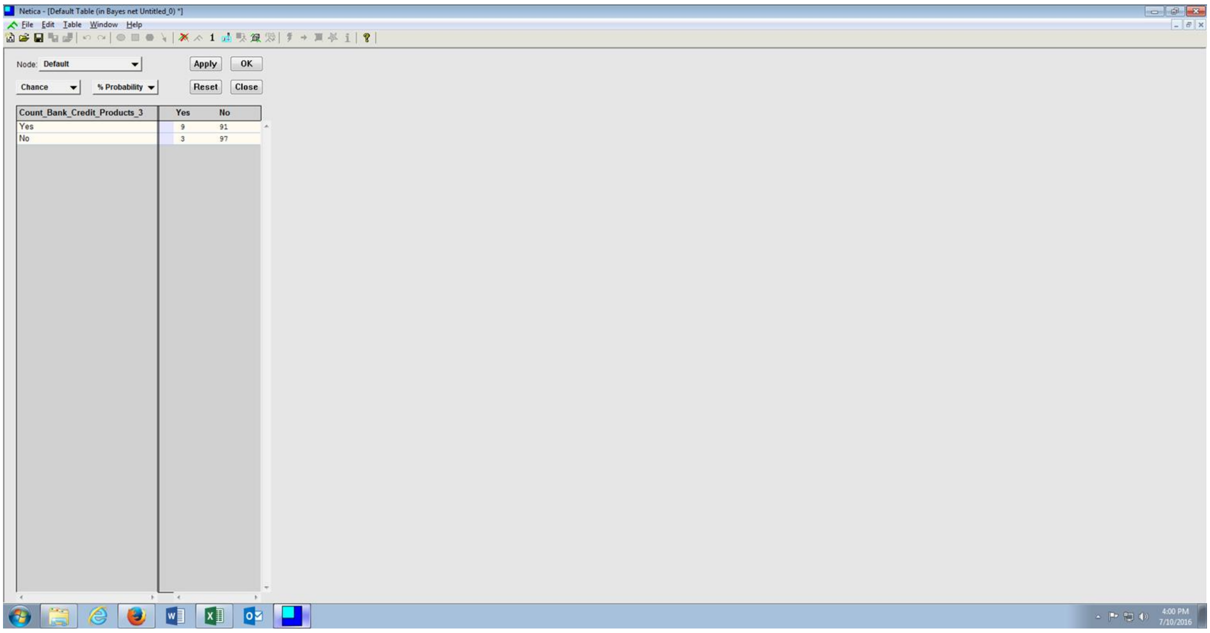
- If Count Bank Card Products > 3 then P(Default) = 9%
- If NOT Count Bank Card Products > 3 then P(Default) = 3%

These probabilities would be updated in the corresponding table:

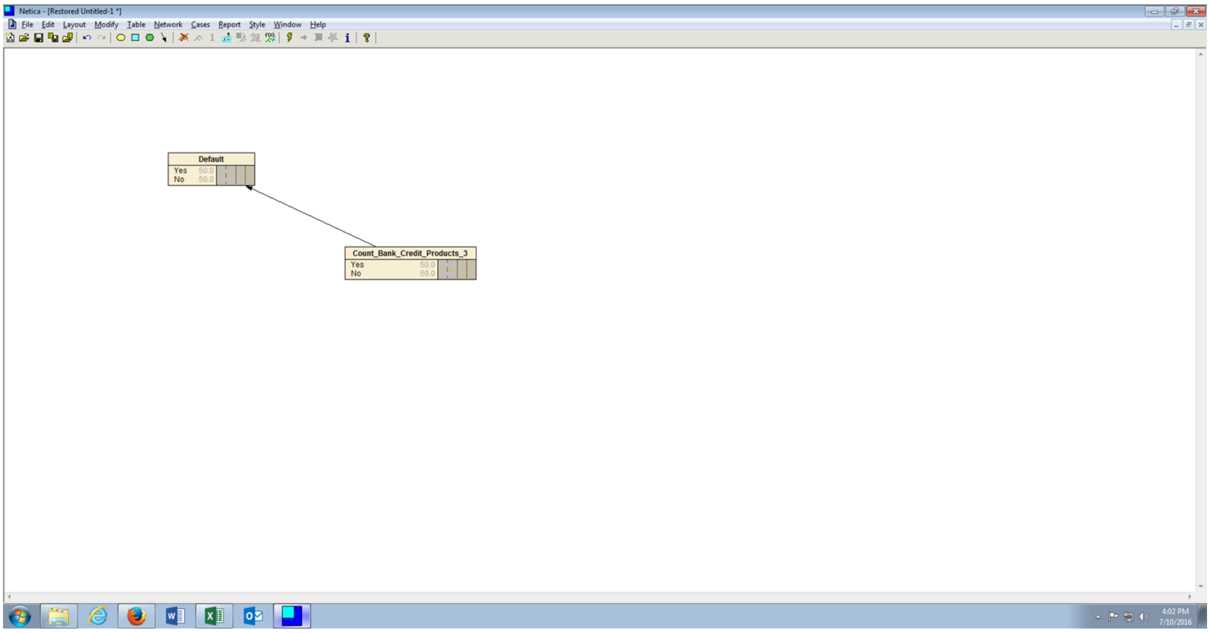


Clicking on the Fill Missing Probabilities Icon will complete the missing probabilities where possible, summing to 100%:

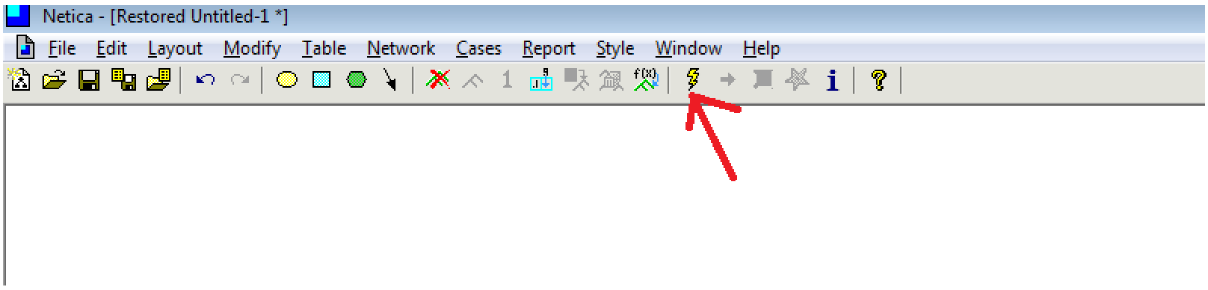


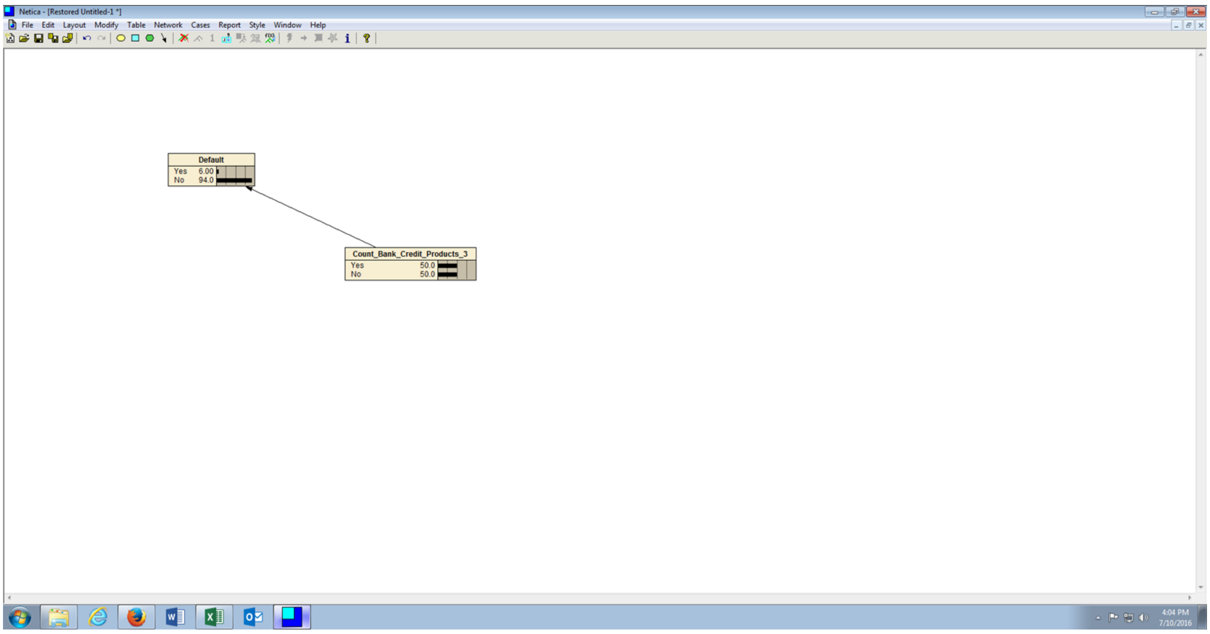


Click Apply, then Ok to close the window. The node probabilities have been set, however the network has not been compiled, and so the states retain the default probabilities:



To compile the network, click on the lightning bolt icon in the menu to compile the network and set the probabilities:

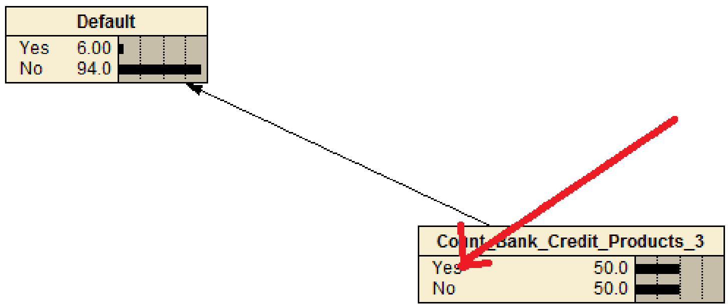




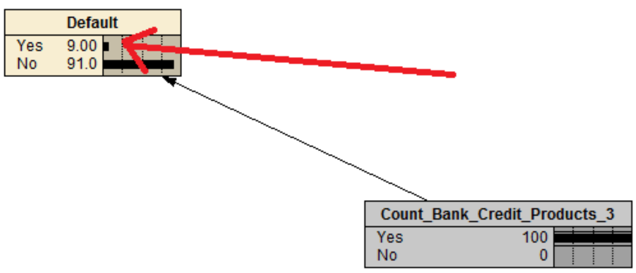
The Bayesian network has now been compiled and is ready to both predict Default and explain Default via Bayesian Inference.

Procedure 5: Manually setting node states to predict and explain.

To make a prediction, which in is in reality a simple matter of recalling the states from the Node \ Conditional Probability tables that were manually entered, it is a simply matter of hovering over the node and state to set, then clicking to set that node:

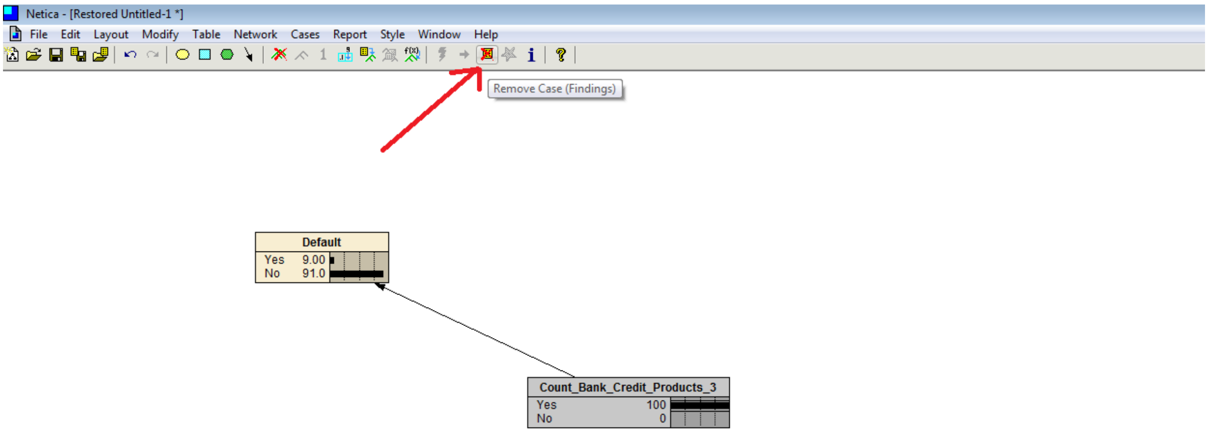


In this example the prediction of whether an account will default is based on the customer having more than three credit products, rather Count_Bank_Credit_Products_3, Yes:

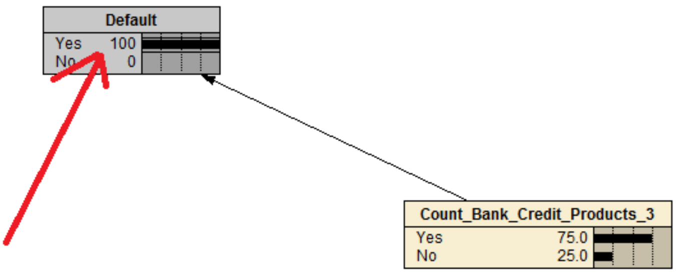


A lookup from the Node \ Conditional probability takes place, in effect, predicting the probability of default to be 9% based on this finding. Forward wise this is an unremarkable prediction based entirely on belief, however Bayesian can perform inference for the purposes of providing explanatory value for the most probable environment surrounding a customer defaulting.

Reset all case findings by clicking the Icon of the same name in the menu:



In this example, click on the Yes state of the Default Node, to update the causation nodes to using Bayesian inference, so to provide some explanatory value as to the environment that causes a customer to default:



In this example it can be observed that a customer is in all probability going to have more than three credit products, if they default.

Procedure 6: Netica Discretisation of Continuous Variable.

Bayesian Methods should be considered as being incompatible with continuous variables as the premise of the analysis technique is that it apportions probability to states (akin to the sides of a dice). Embracing the state only maxim of Bayesian Networks, presented with a continuous variable, the task is to convert that continuous variable into a state.

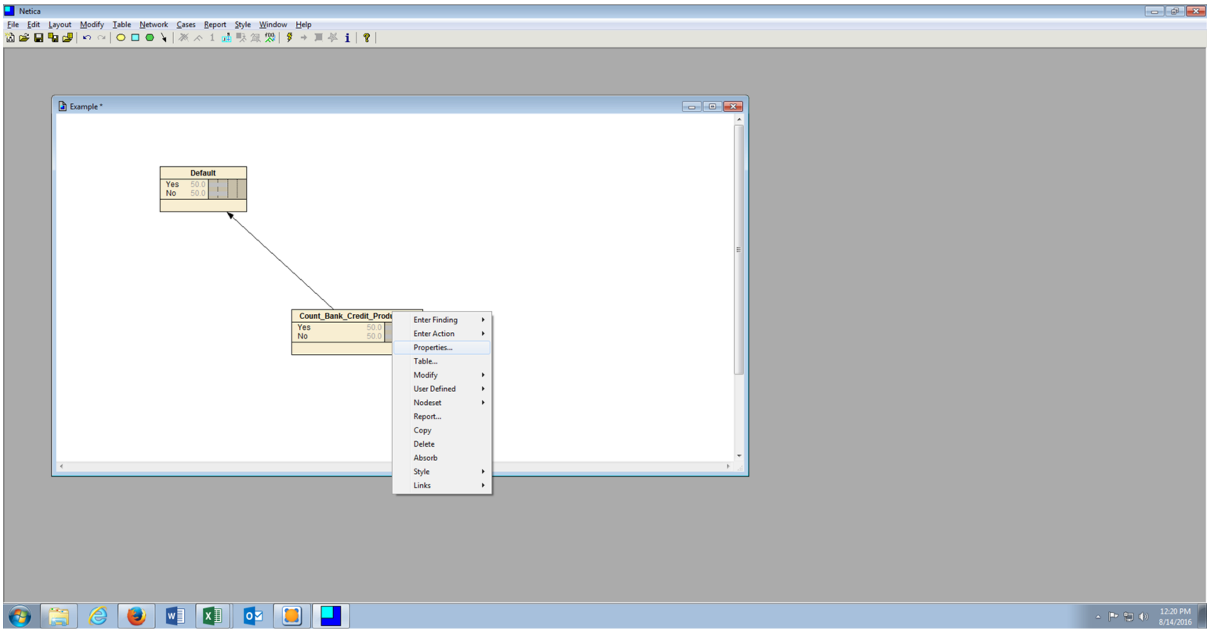
In the procedures thus far there have been several methods presented to bin variables for the purposes of model improvement (reference procedure 12). Netica provides a quick and convenient means to turn continuous variables into states, a process it refers to as discretisation.

There are three useful automated forms of discretisation offered by Netics:

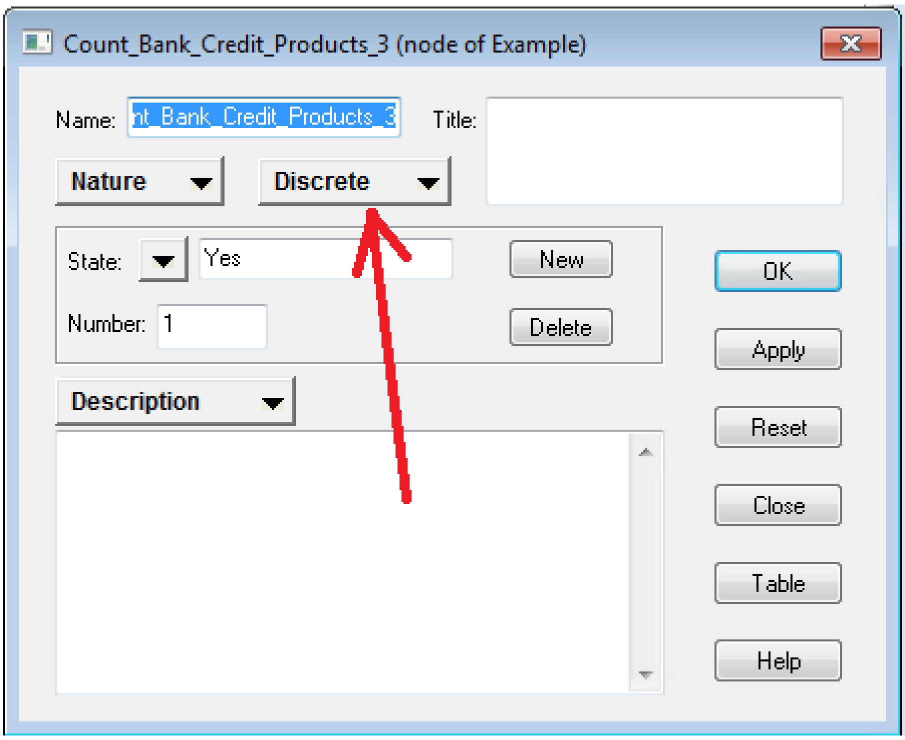
- Fixed Bin
- Exponential Bin
- Natural Logarithm

The boundaries can be bound by -infinity or infinity if it is felt that the lower or upper bounds may change over time.

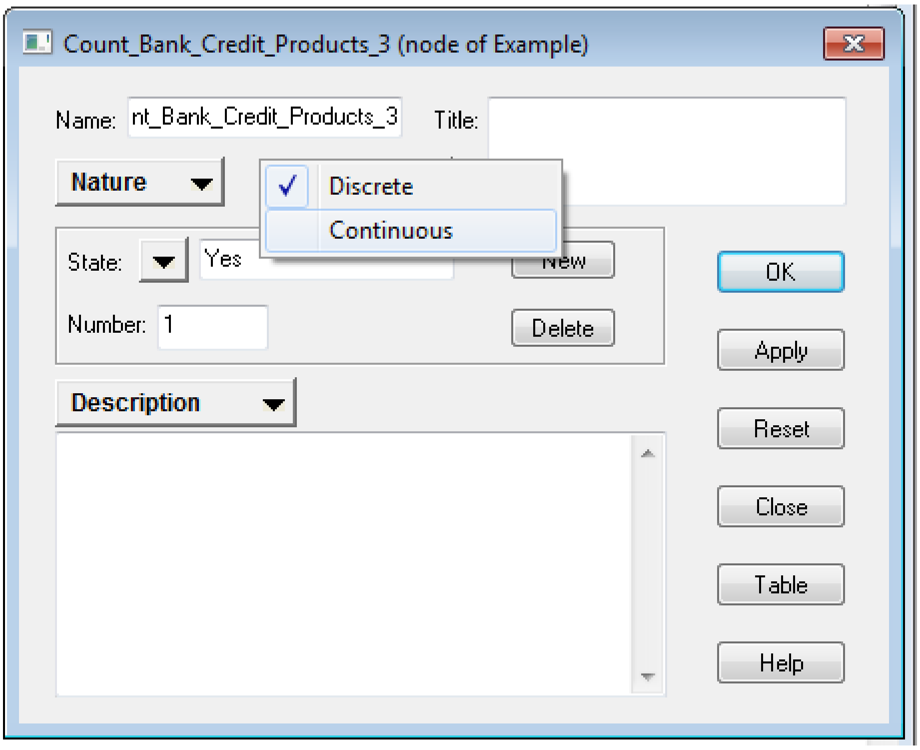
To enter the discretisation for a Node, right click on the node, then click properties:



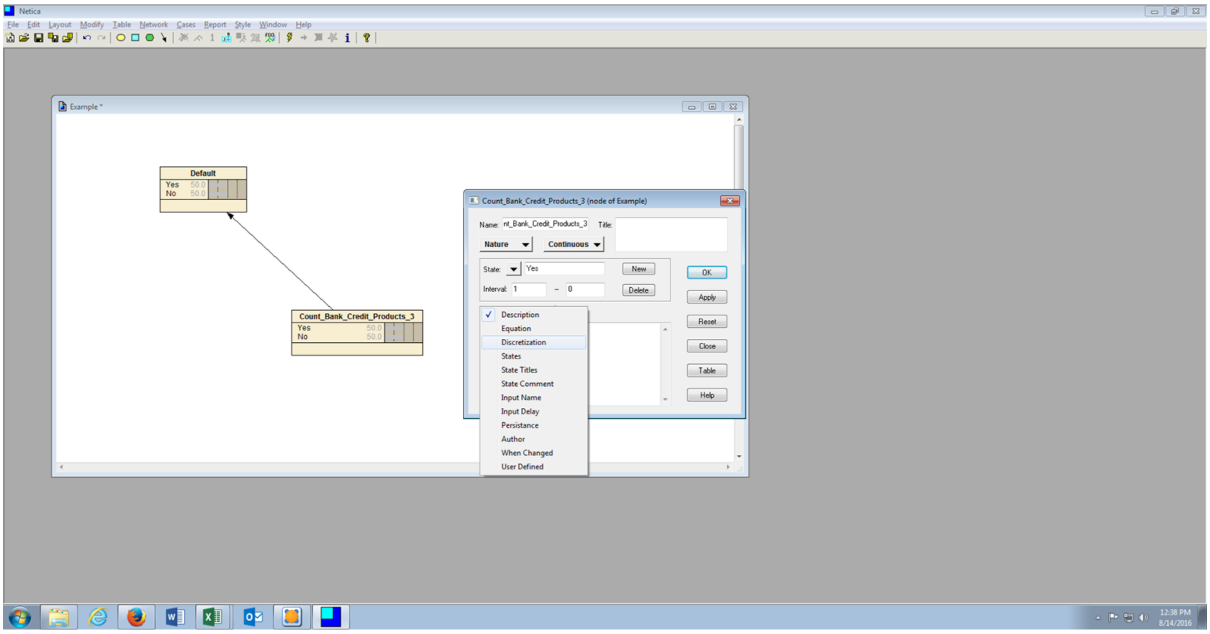
It can be noted that the current node is set as Discrete, which means that States and their values are entered manually:



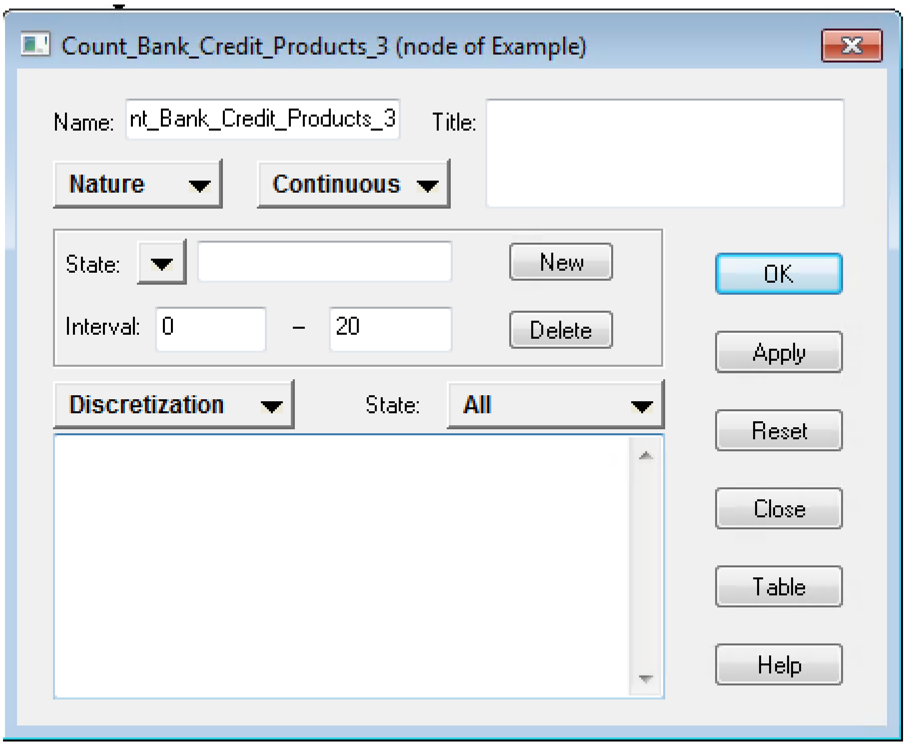
Click on the button Discrete which will present the opportunity to change the node to be Continuous:



Upon changing the node type to Continuous, click on the Description button which will expose a sub menu, then select Discretisation:

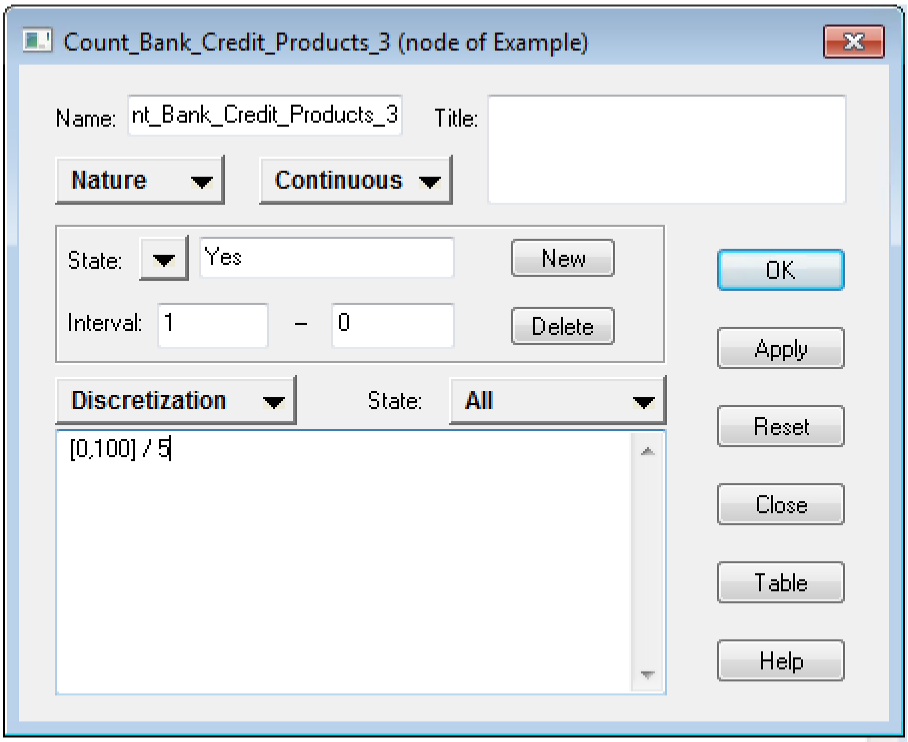


On clicking the Discretisation button, the large textbox will now accept (rather process) the shorthand notation that will divide a continuous variable into states:

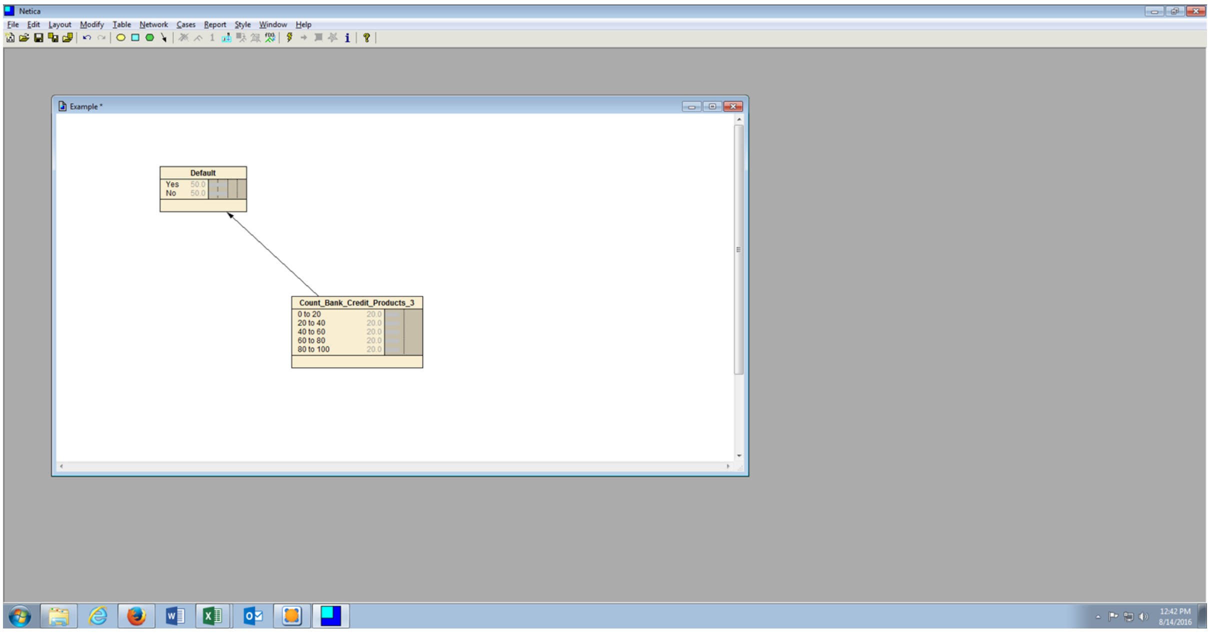


Clearing out any existing values, shorthand will be used to specify the lower boundary, the upper boundary and the number of bins between these boundaries, in this example 0 is the lower boundary, 100 is the upper boundary and there are to be 5 bins:

[0,100] / 5



Upon clicking OK the node will be updated with these states. If prompted to remove existing states, click OK:



This example uses a Fixed Bin shorthand. There are three types of shorthand available, where the values in highlight are the parameters:

- Fixed Bin (as example): [Begin,End] / Bin
- Exponential Bin: [Begin, End] +%Bigger
- Natural Logarithm: [Begin, End] / L Bin

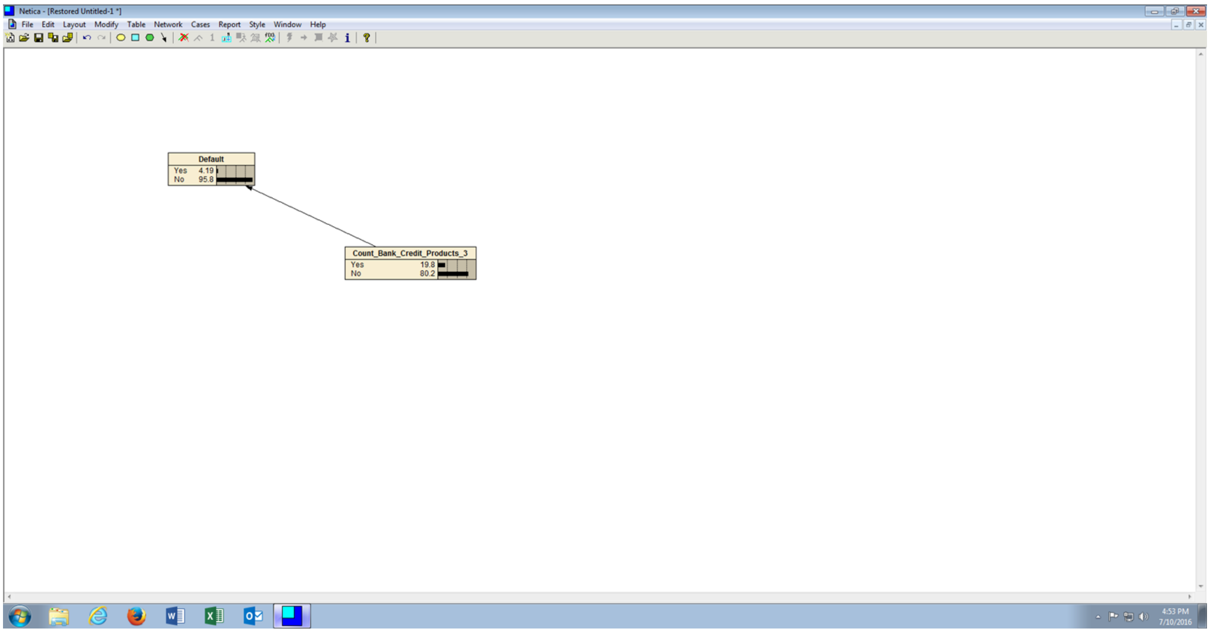
If the production values of the upper and lower bound are not known at design time, then -infinity or infinity can be used as lower and upper bound respectively. The use of infinity will bring about runtime resizing of the bounds.

Procedure 7: Learn node probabilities.

Up to this point the procedures have created a naive Bayesian network based on belief, belief being an encapsulation of subjective probability in Node \ Conditional probability tables.

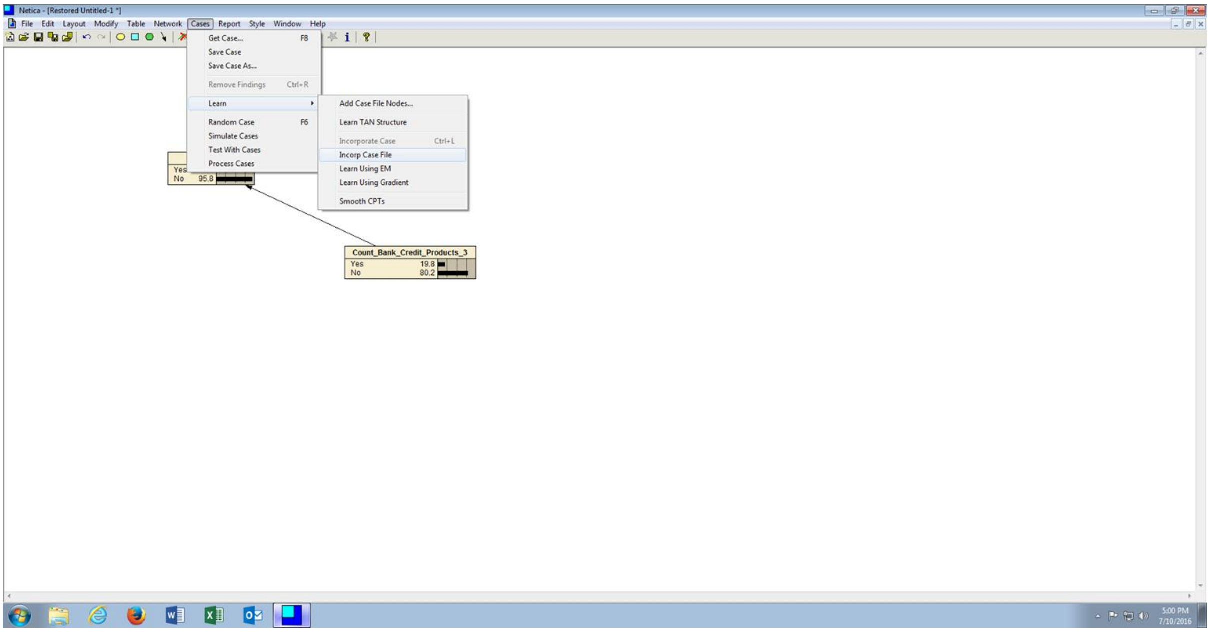
Subjective probability is extremely good when derived in a group and can allow for the creation of predictive analytics models where there is no data available (another tool for such scenarios is conjoined Regression \ ANN). In the event that data is available, it is far better to train the structure with real probabilities based upon the contents of a data file.

The procedure to train a Bayesian network is quite simple. Start by resetting all findings, as specified in procedure 39, then clicking into the canvas to ensure that no node is selected:

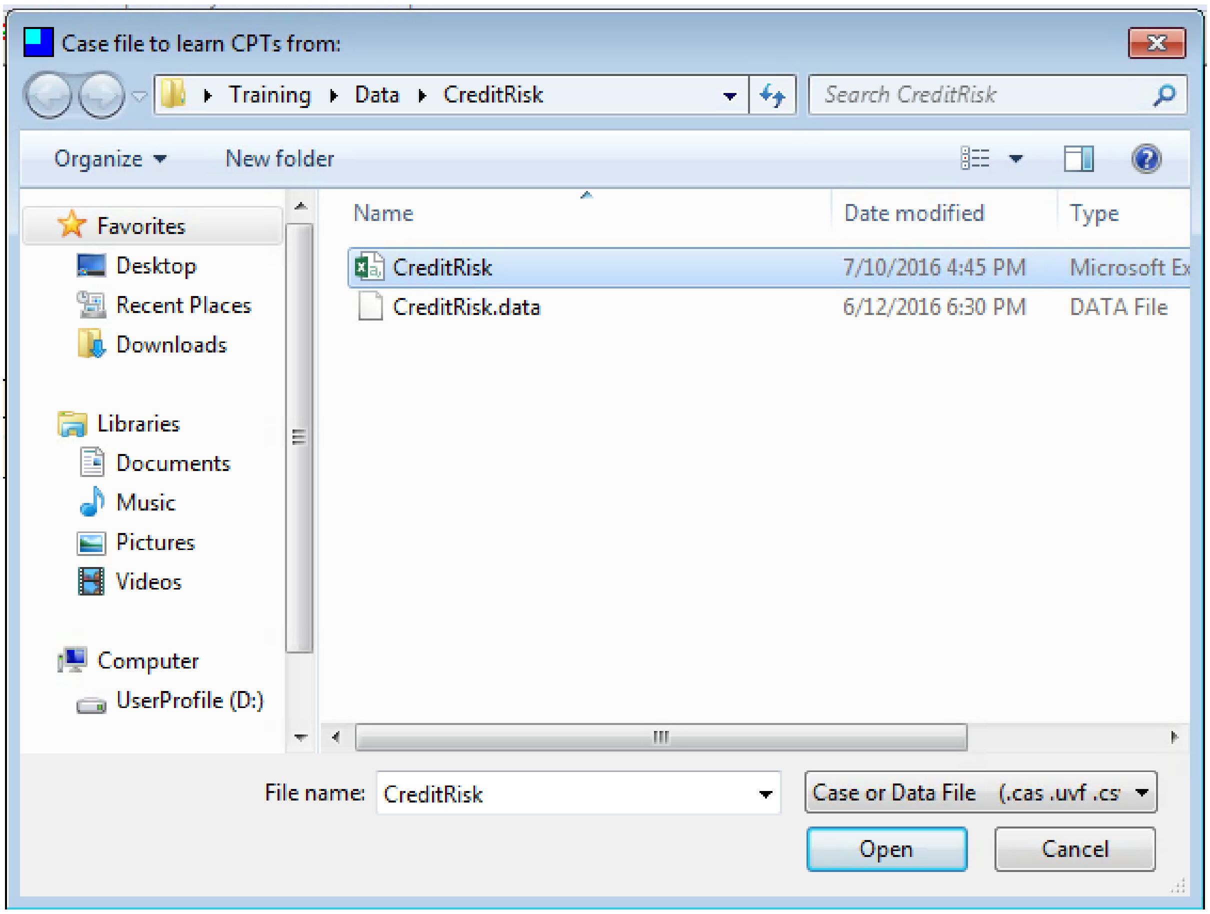


It is very important that the name of the nodes match the names of the columns in the file that is intended to train the Bayesian Network and that all of the states that exist in the data, are reflected in the respective nodes.

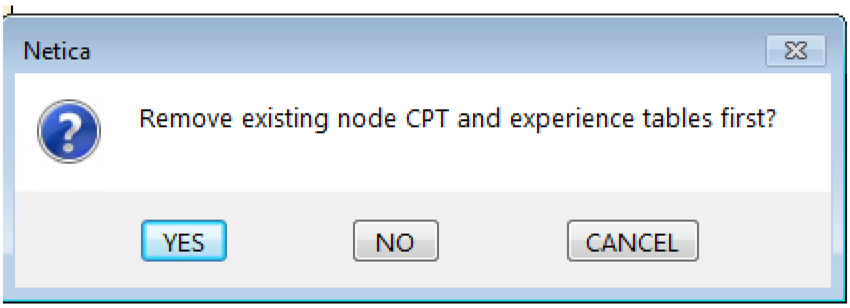
To train the Bayesian Network, click on the menu item Cases, then click or hover on the Learn sub menu item, then click Incorporate Case File (Learn using EM achieves the same but is better where data is thought to be missing):



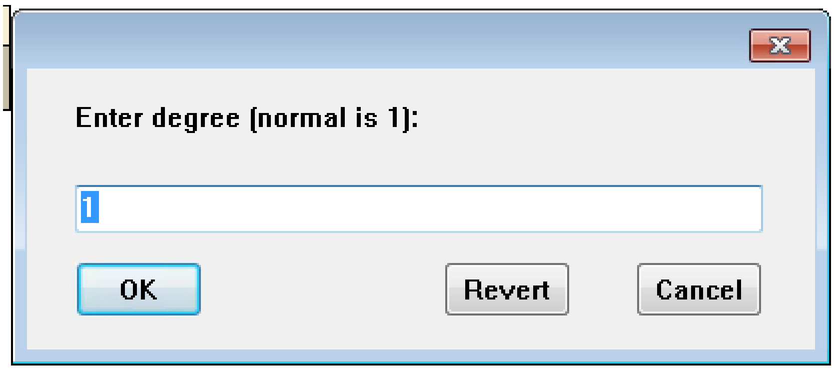
Locate the file to be used for training, in this case CreditRisk.csv:



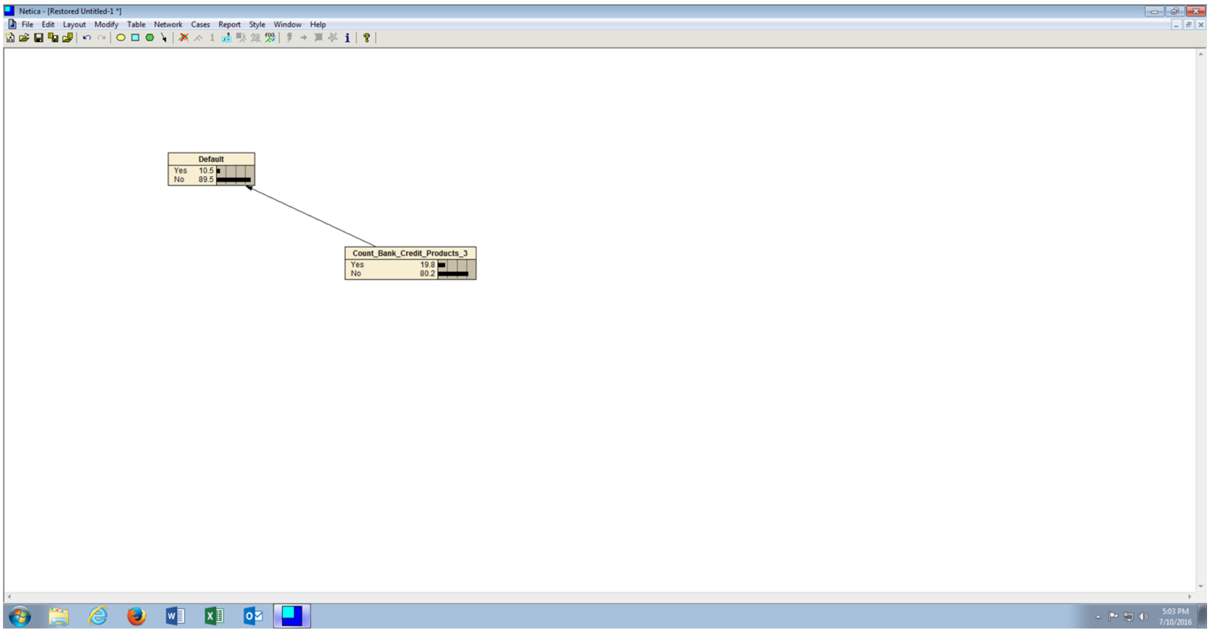
Click open once the CreditRisk.csv file has been identified to begin the training process. Remove pre-existing Node \ Conditional probability tables if prompted to do so:



Maintain the default degree of 1 when prompted:



The network has now been trained using actual probabilities identified in the data rather than those added subjectively:



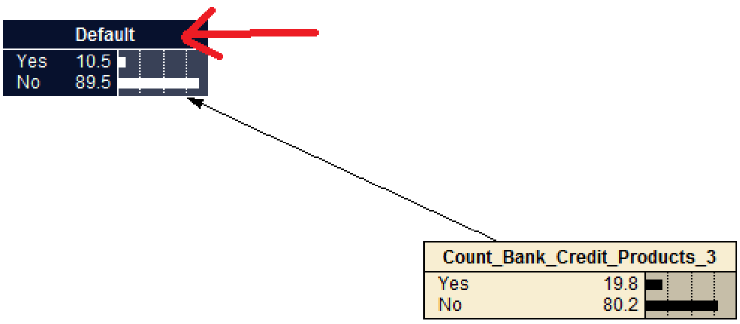
An interesting exercise is to observe the difference between subjective and frequentist (i.e. learned) probabilities.

Procedure 8: Test Classification Accuracy of a Bayesian Network.

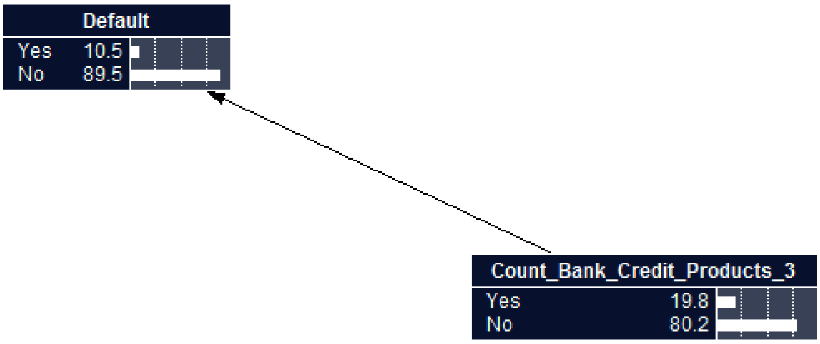
Bayesian Networks are viewed to be extremely useful for classification problems with the measure of the performance of being classification accuracy, commonly presented as a confusion matrix (in the same manner as Logistic Regression).

Bayesian networks, once constructed and trained, can facilitate a testing process which produces similar analysis to that observed in logistic regression procedures.

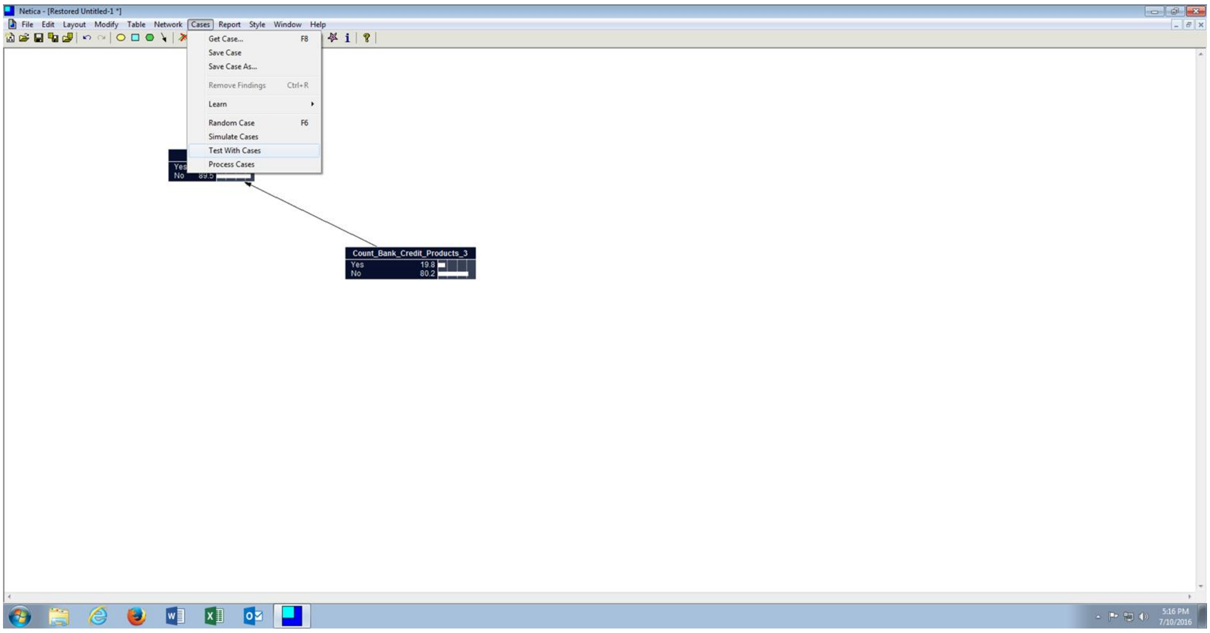
Firstly, highlight all nodes required by holding down the ctrl key and clicking the node name:



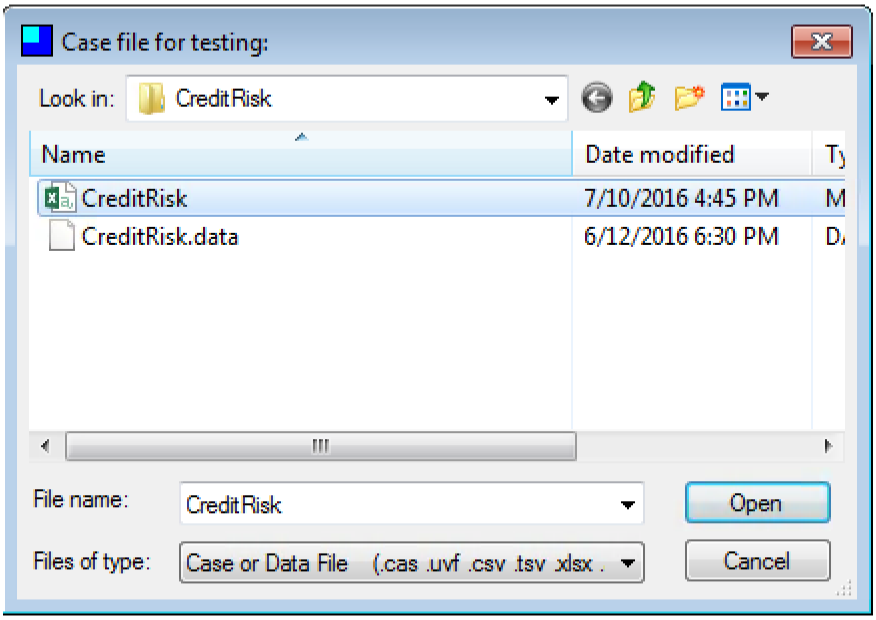
JUBE



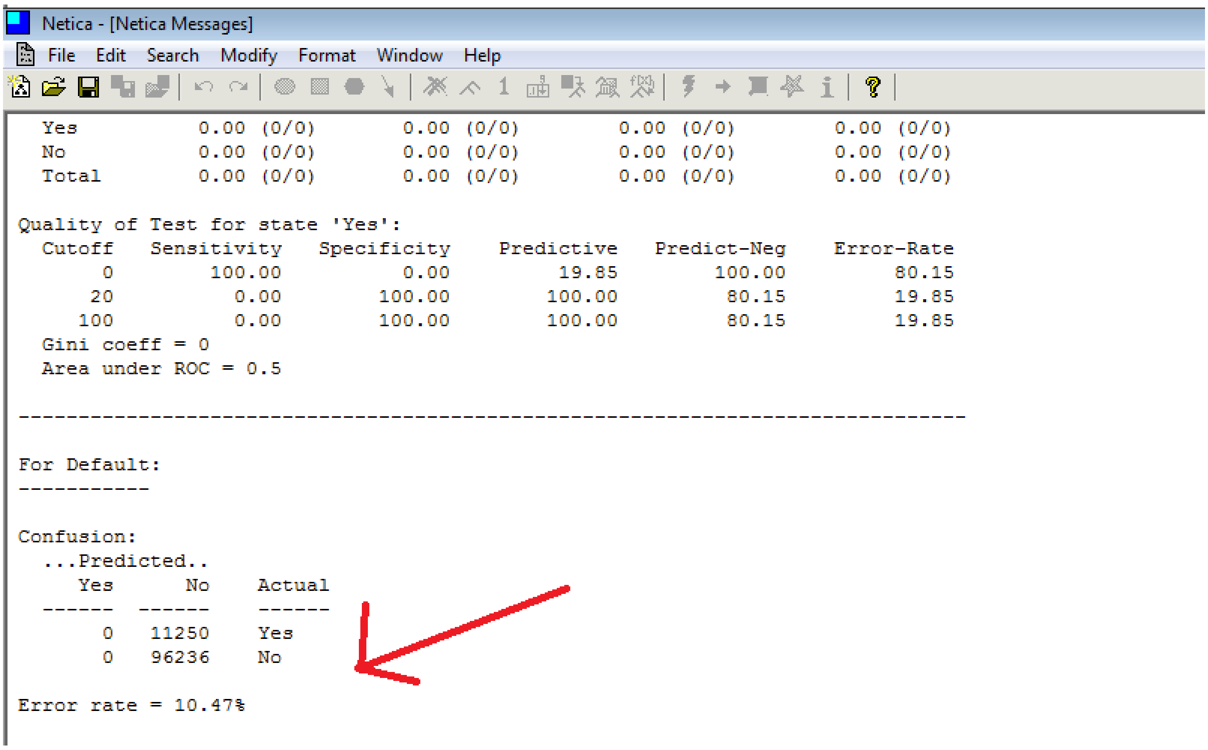
To test the network, navigate to the Cases menu, then click on the Test with Cases sub menu:



Select the CreditRisk.csv file when prompted to open a file:



Clicking the Open button begins the testing process, for the dependent variable, this is Default in this example, a Confusion Matrix and Error Rate is presented, being the main focus of optimisation in a stepwise approach, or perhaps using more automated means to add nodes to the canvas and establish relationships between the independent variables:

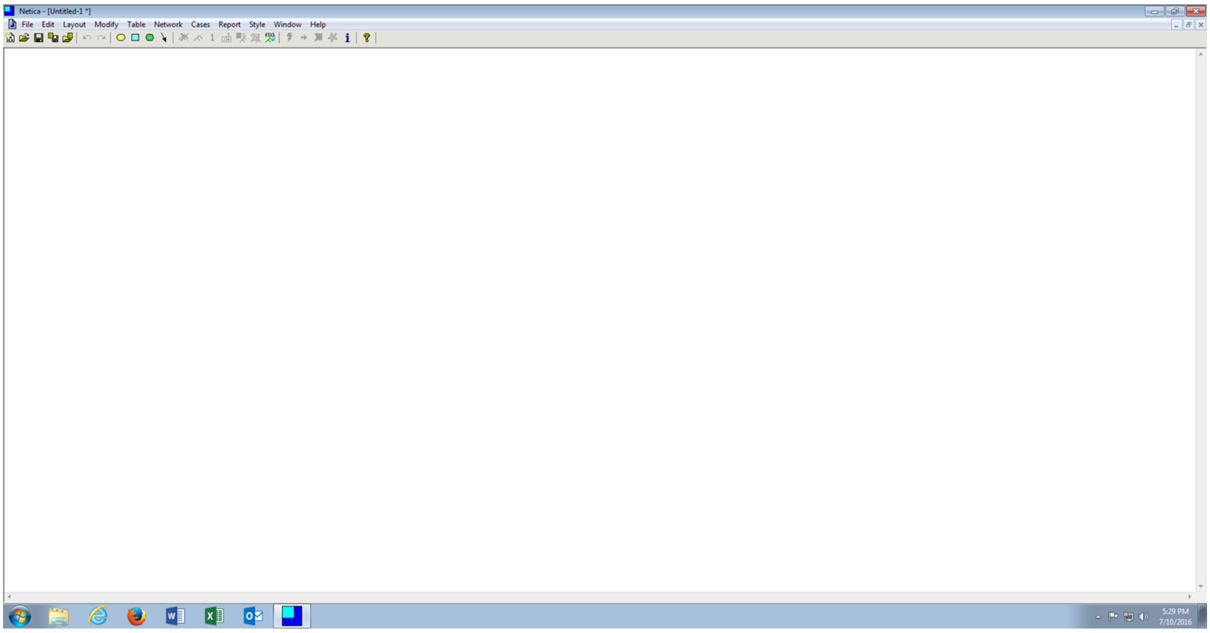


Procedure 9: Add Nodes Automatically to a Canvas.

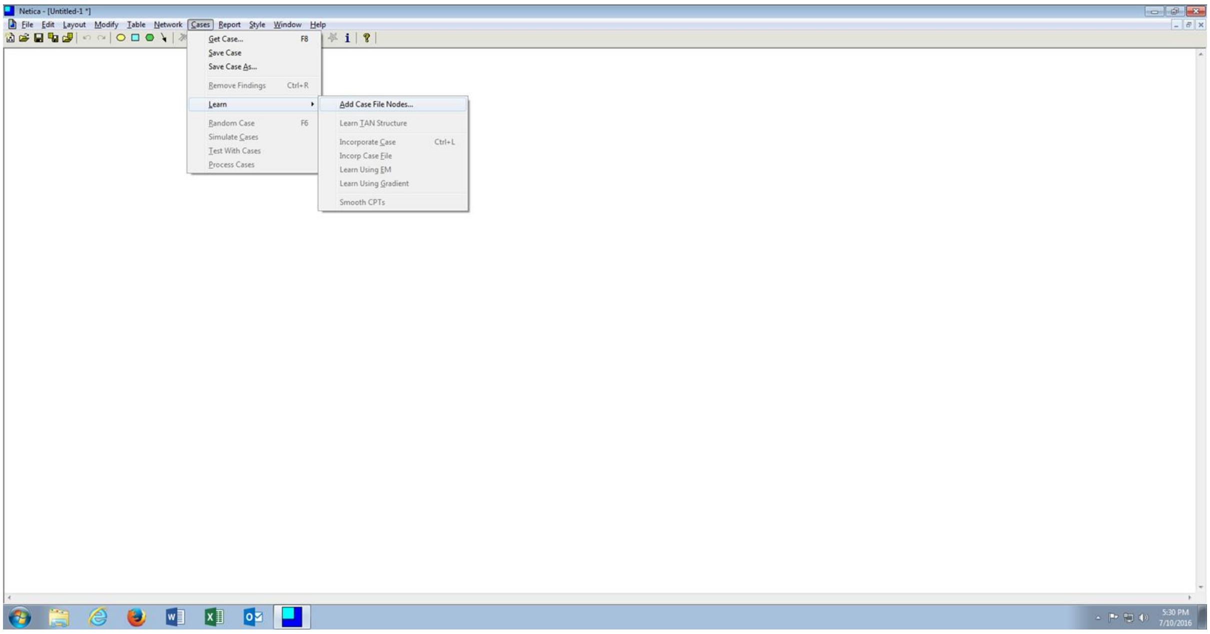
The process of manually adding nodes to a canvas is quite laborious and with a key benefit of Bayesian networks being the ability to handle extremely large networks with hundreds of nodes, impractical. Furthermore, Bayesian techniques are inherently state based, which would rely on a process of dividing continuous variables into appropriate state bins.

Netica has the ability to infer columns from a file, thus allowing for automation in the creation of nodes on the canvas.

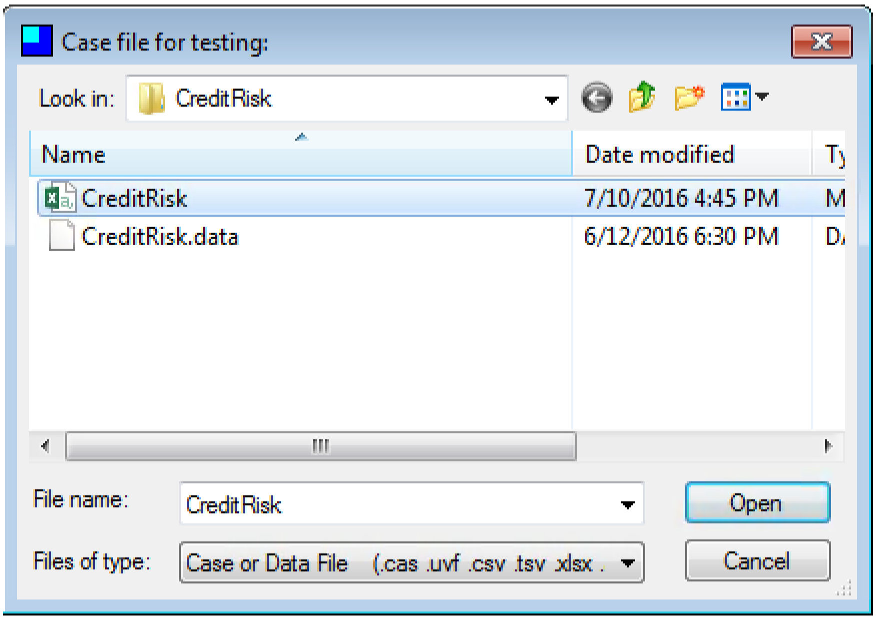
Start by creating a blank canvas as demonstrated in procedure 35:



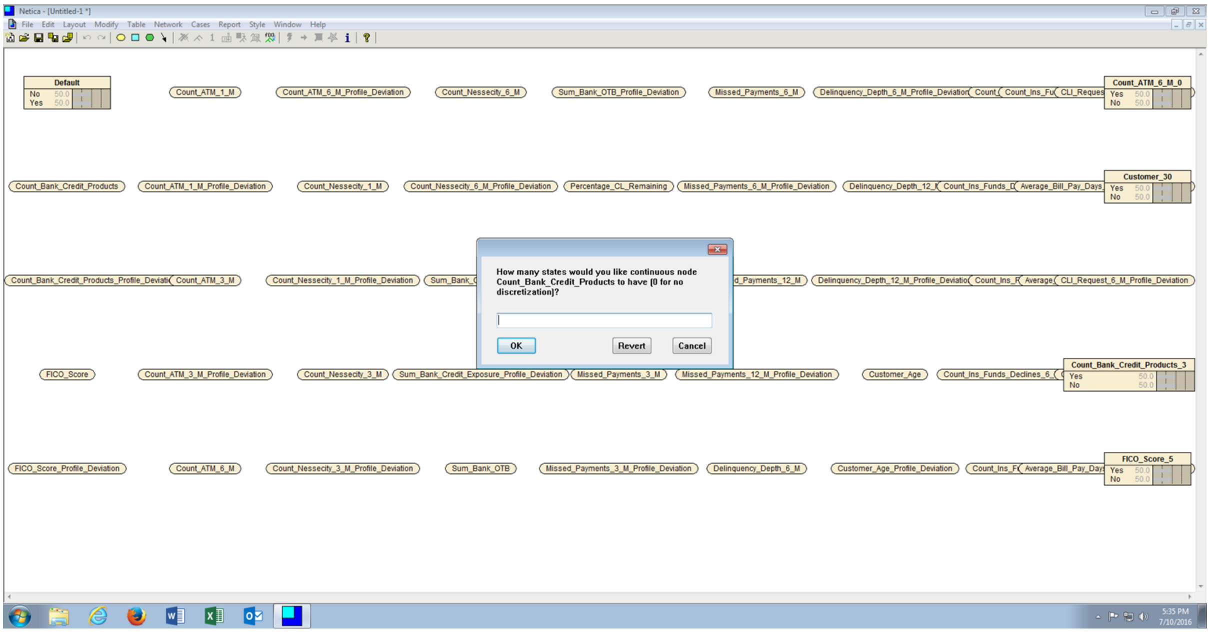
To infer and then add the case file nodes, click Cases in the menu Item, then click or hover on the Learn sub menu item, then click Add Case File Nodes:



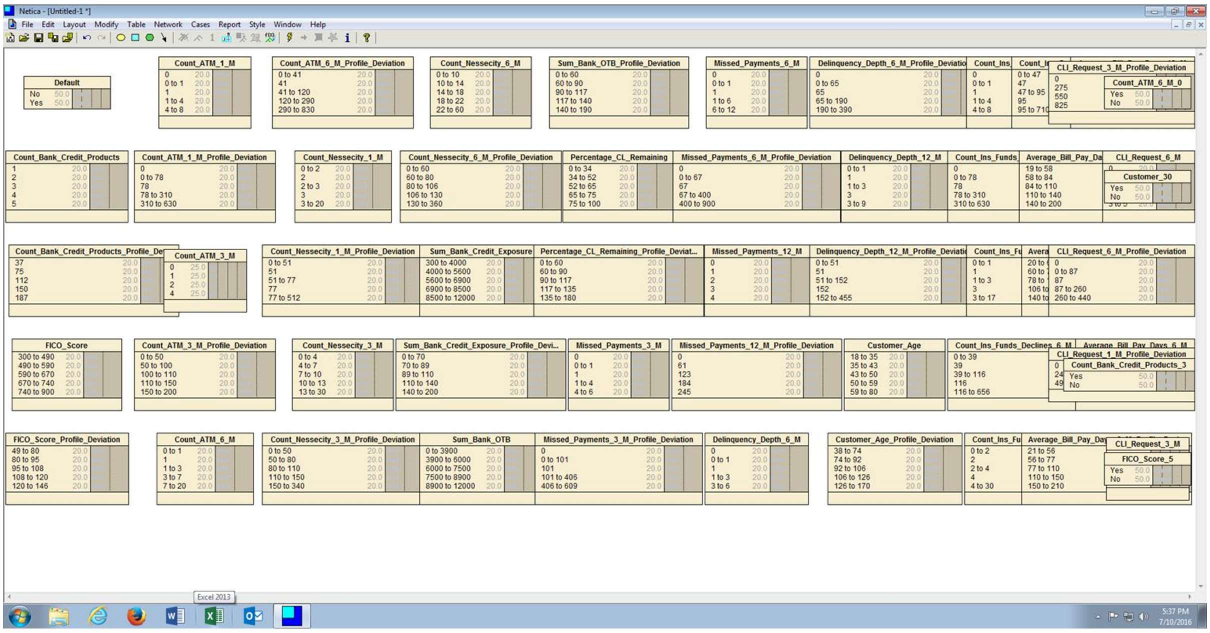
When the dialog box opens, select the file CreditRisk.csv:



Clicking Open will begin the process of creating nodes based upon the Variables name coupled with an analysis of the number of states within that Variable. In the event that a variable is determined to be continuous, a prompt will be displayed to determine the number of states to set for this variable:



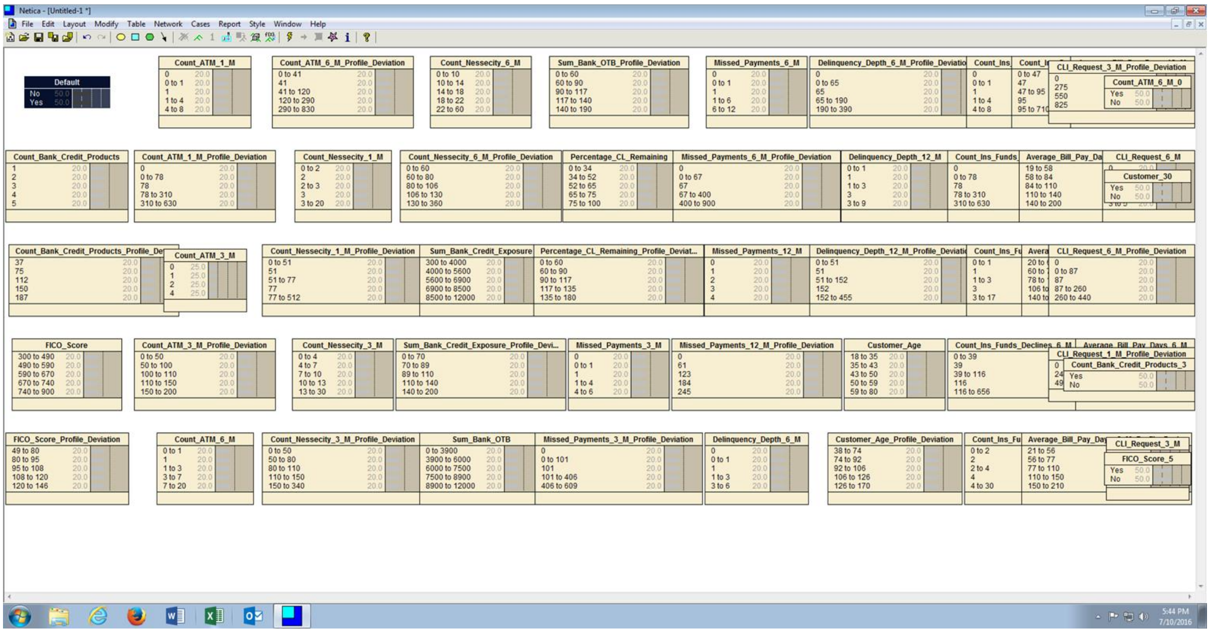
Specify the number of states deemed appropriate for the variable, then click ok. Repeat for each variable until all of the nodes have been added to the canvas:



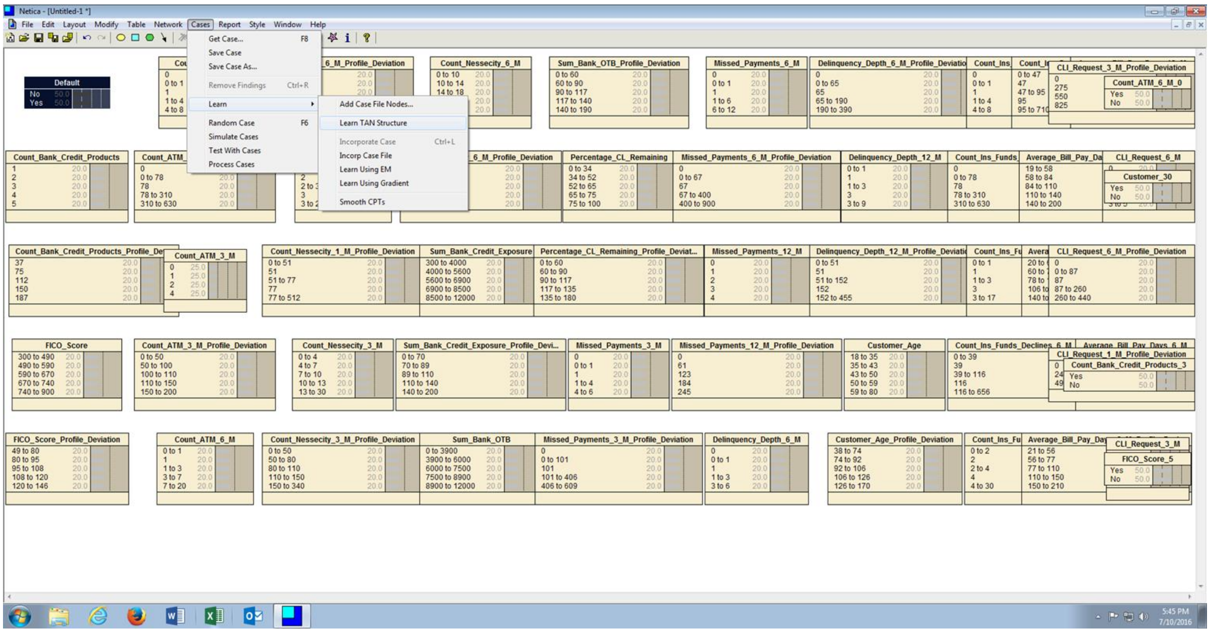
Procedure 11: Learn TAN Structure to Link Nodes Automatically.

When dealing with an overwhelming number of nodes it is possible to automatically link these nodes, based firstly on a naive structure similar to the that manually created in procedure 37, then augmenting this structure to look for relationships between the nodes that may be of interest. This approach is called Tree Augmented Naïve Bayesian Networks, or TAN Bayesian Networks.

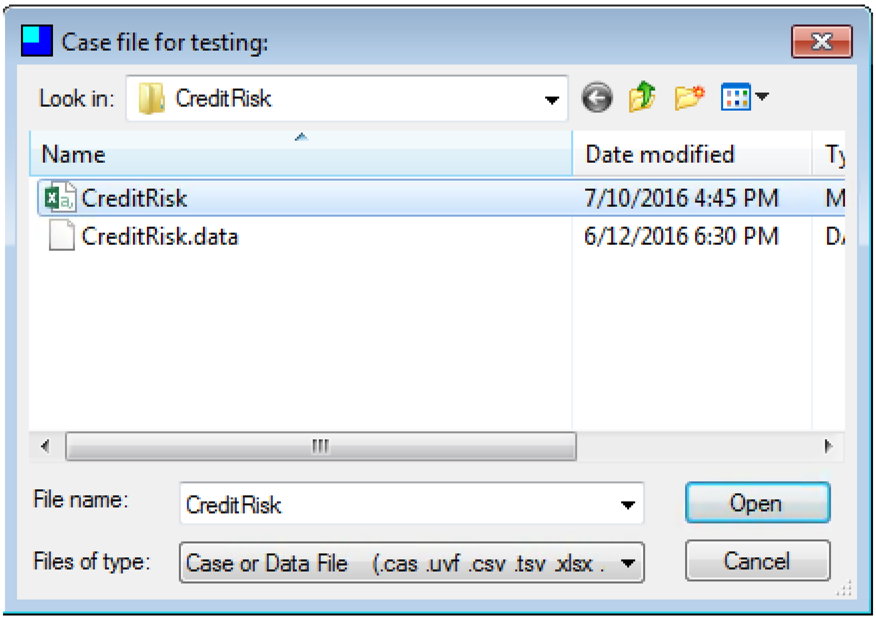
For Netica to learn the structure, start by selecting the dependent variable node in the canvas, in this case Default:



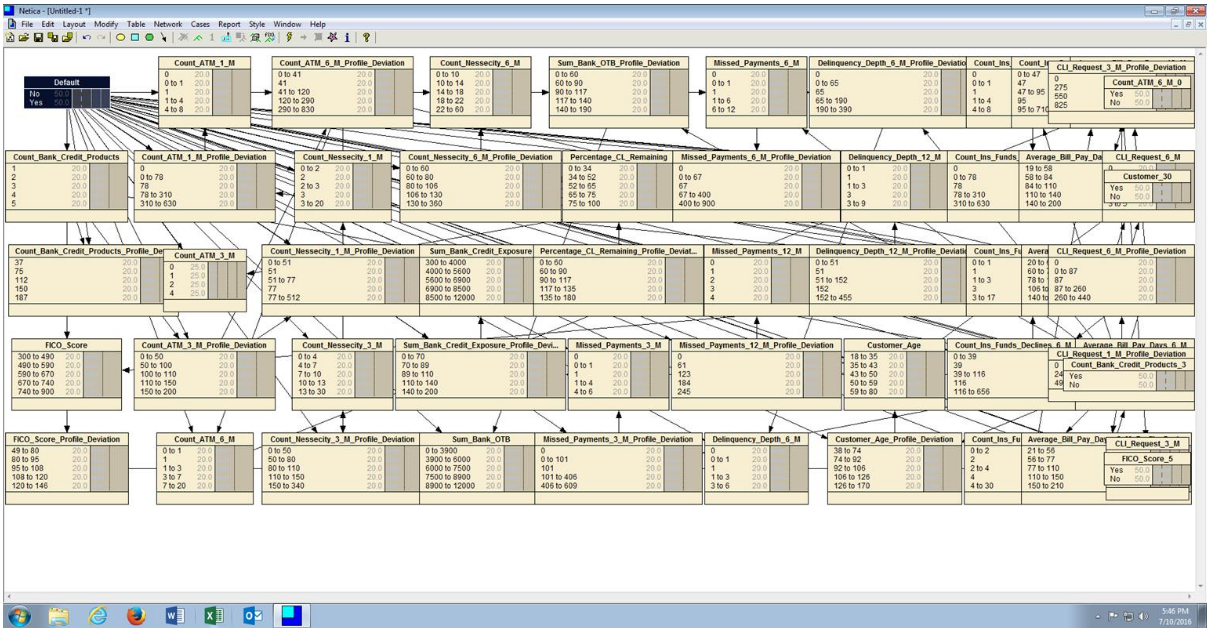
To learn the structure using Tre Augmented Naive approaches, click on the Cases menu item, click or hover on Learn, then click Learn TAN Structure:



Select the file to be used for the purposes of training the structure, in this example CreditRisk.csv:



Clicking OK will begin the learning process:



Firstly, all nodes will be linked to / from the dependent variable, thereafter relationships to / from independent variables will be established.

The direction of the link is not that important as Bayesian Inference will be performed, however if links do not follow the direction of causation, maintaining node \ conditional probability table can become bewildering. It follows that a learnt TAN structure would likely be used only where probabilities are going to be learnt also. It follows that the execution of procedure 40 should occur to update the node probability tables, followed by procedure 41 to determine the classification accuracy of this network, so to determine if this extremely complex network provides any uplift on a simpler network.

Module 13: Neural Networks

Neural networks are a universal predictive analytics method, if a little unexplainable when they grow large. Unlike many of the predictive analytics techniques presented, Neural Networks are as equally good at classification problems as they are prediction problems. This module will use dataset used in procedures 90 and 93, seeking to showcase the improvement that can be obtained in using Neural Networks, albeit with an increase in complexity and explainability.

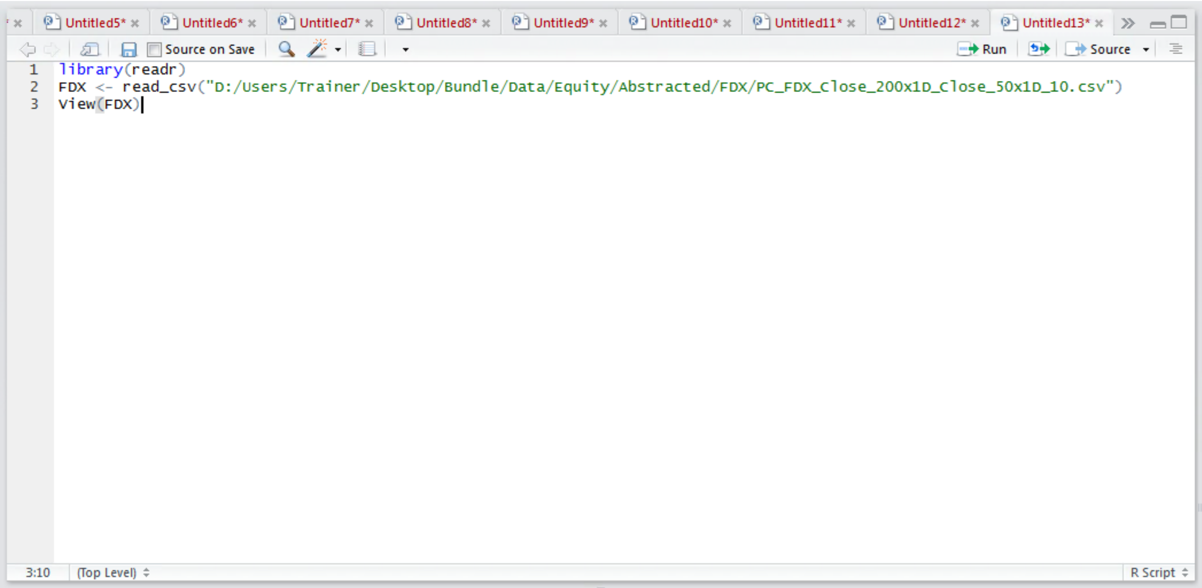
Procedure 1: Train a Neural Network.

In this procedure, improvement will be sought from module 6, using the FDX dataset. Start by importing the dataset using the readr package and read.csv() function (as there are no strings to be converted to factors):

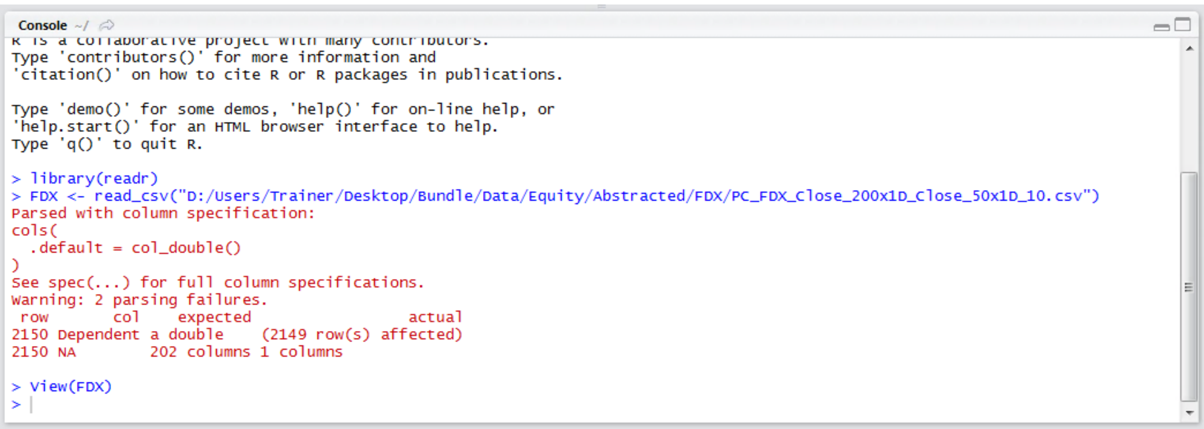
library(readr)

FDX <-
read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")

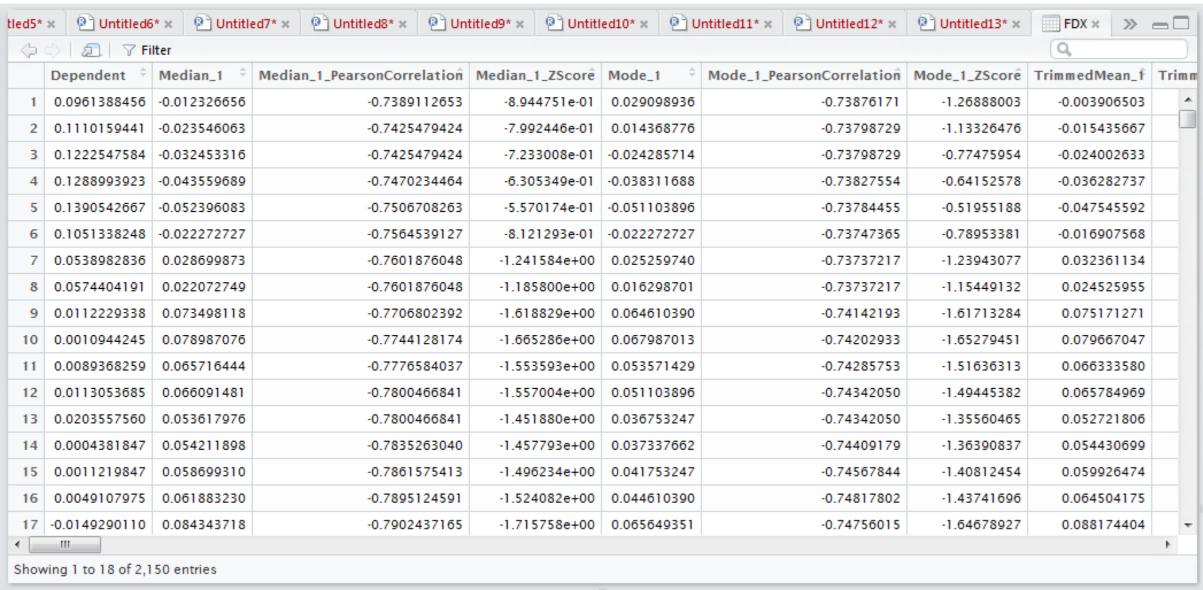
View(FDX)



Run the line of script to console:



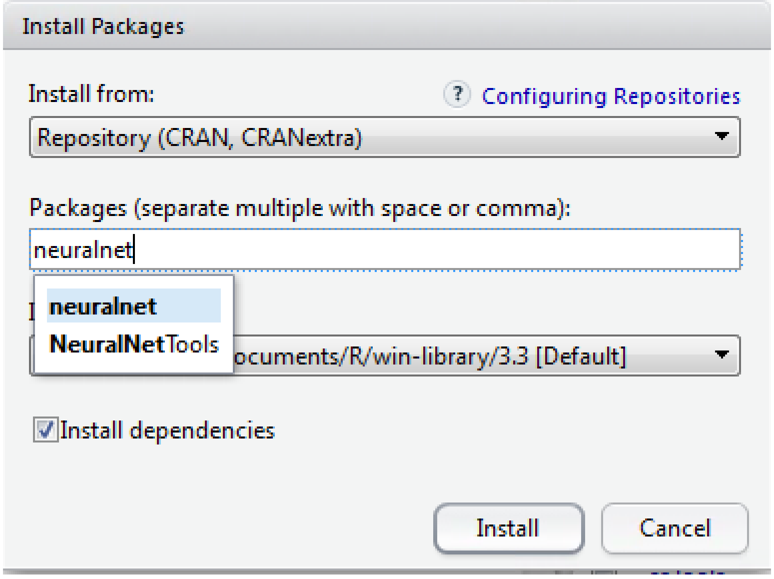
It can be seen via the RStudio viewer that the FDX dataset has been loaded into R:



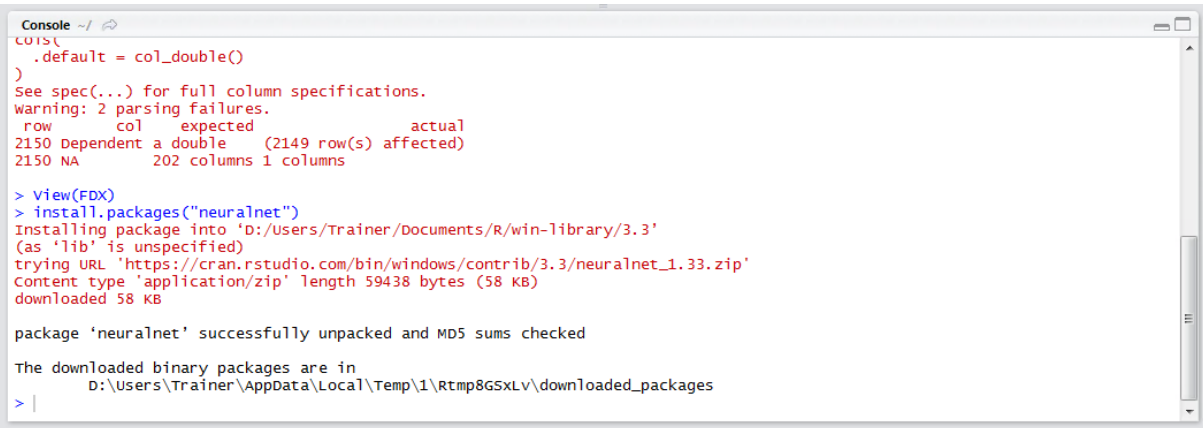
	Dependent	Median_1	Median_1_PearsonCorrelation	Median_1_ZScore	Mode_1	Mode_1_PearsonCorrelation	Mode_1_ZScore	TrimmedMean_1
1	0.0961388456	-0.012326656	-0.7389112653	-8.944751e-01	0.029098936	-0.73876171	-1.26888003	-0.003906503
2	0.1110159441	-0.023546063	-0.7425479424	-7.992446e-01	0.014368776	-0.73798729	-1.13326476	-0.015435667
3	0.1222547584	-0.032453316	-0.7425479424	-7.233008e-01	-0.024285714	-0.73798729	-0.77475954	-0.024002633
4	0.1288993923	-0.043559689	-0.7470234464	-6.305349e-01	-0.038311688	-0.73827554	-0.64152578	-0.036282737
5	0.1390542667	-0.052396083	-0.7506708263	-5.570174e-01	-0.051103896	-0.73784455	-0.51955188	-0.047545592
6	0.1051338248	-0.022272727	-0.7564539127	-8.121293e-01	-0.022272727	-0.73747365	-0.78953381	-0.016907568
7	0.0538982836	0.028699873	-0.7601876048	-1.241584e+00	0.025259740	-0.73737217	-1.23943077	0.032361134
8	0.0574404191	0.022072749	-0.7601876048	-1.185800e+00	0.016298701	-0.73737217	-1.15449132	0.024525955
9	0.0112229338	0.073498118	-0.7706802392	-1.618829e+00	0.064610390	-0.74142193	-1.61713284	0.075171271
10	0.0010944245	0.078987076	-0.7744128174	-1.665286e+00	0.067987013	-0.74202933	-1.65279451	0.079667047
11	0.0089368259	0.065716444	-0.7776584037	-1.553593e+00	0.053571429	-0.74285753	-1.51636313	0.066333580
12	0.0113053685	0.066091481	-0.7800466841	-1.557004e+00	0.051103896	-0.74342050	-1.49445382	0.065784969
13	0.0203557560	0.053617976	-0.7800466841	-1.451880e+00	0.036753247	-0.74342050	-1.35560465	0.052721806
14	0.0004381847	0.054211898	-0.7835263040	-1.457793e+00	0.037337662	-0.74409179	-1.36390837	0.054430699
15	0.0011219847	0.058699310	-0.7861575413	-1.496234e+00	0.041753247	-0.74567844	-1.40812454	0.059926474
16	0.0049107975	0.061883230	-0.7895124591	-1.524082e+00	0.044610390	-0.74817802	-1.43741696	0.064504175
17	-0.0149290110	0.084343718	-0.7902437165	-1.715758e+00	0.065649351	-0.74756015	-1.64678927	0.088174404

Showing 1 to 18 of 2,150 entries

To train a neural network, firstly download and install the package using the RStudio interface:

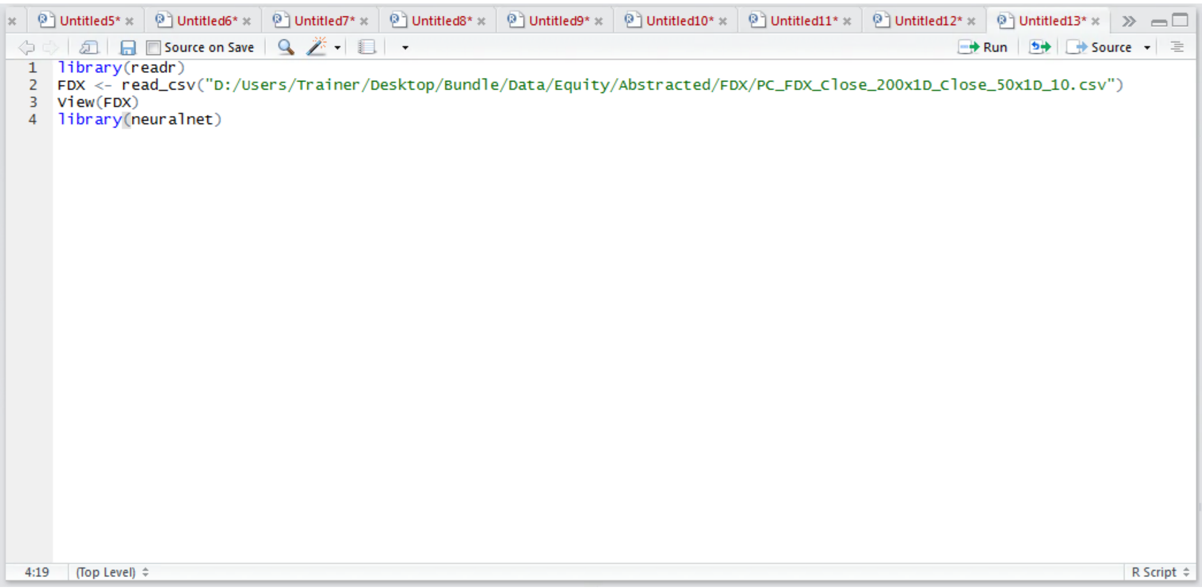


Click Install to execute the installation:



Load the library:

library(neuralnet)



Run the line of script to console:

```
Console ~/
See spec(...) for full column specifications.
Warning: 2 parsing failures.
 row       col    expected                actual
2150 Dependent a double    (2149 row(s) affected)
2150 NA        202 columns 1 columns

> View(FDX)
> install.packages("neuralnet")
Installing package into 'D:/Users/Trainer/Documents/R/win-library/3.3'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/neuralnet_1.33.zip'
Content type 'application/zip' length 59438 bytes (58 KB)
downloaded 58 KB

package 'neuralnet' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        D:\Users\Trainer\AppData\Local\Temp\1\Rtmp8GSxLv\downloaded_packages
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 3.3.3
> |
```

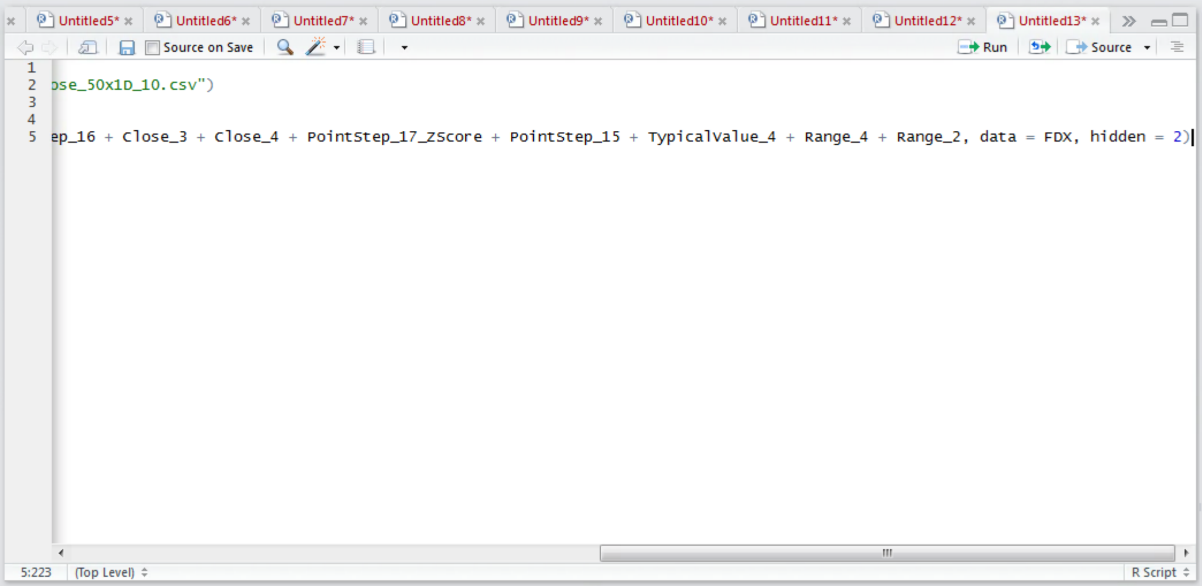
In this example, a warning has been displayed saying that the build was done in a later version of R, however backward compatibility can be reasonably assured and as such the warning can be ignored. Once R version 3.3.3 has become stable, it might be worth upgrading.

Building, or training, a Neural Network is very similar to building a regression model, save for a few parameters nuanced to this function (not least that the overall package is VERY unforgiving with almost no intuitive error messages). In this example, a neural network will be created with an arbitrary four processing elements, with one hidden layer. The dot notation, typically used to instruct all variables, does not work with this function currently (it is a bug) and so a manually constructed formula need be created.

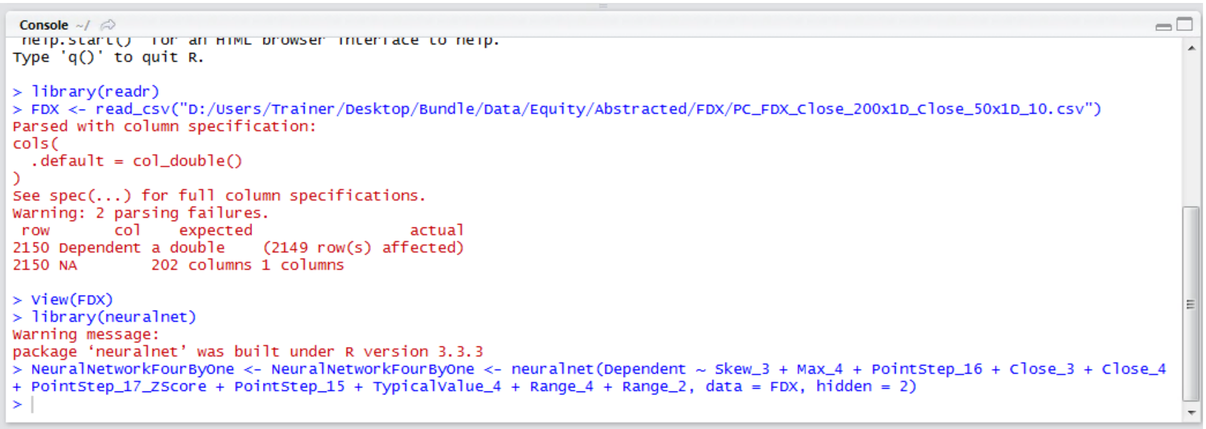
Furthermore, for the purposes of these procedures, it is beneficial to have a slightly more limited feature set owing to the time it would take to train and that, despite popular belief, less is quite often more when training Neural Networks. It is also worth noting that the neuralnetwork() function is a single threaded function and can take a VERY long time to train upon data frames which contain many records and many independent variables.

I this example, a neural network is going to be built upon 10 independent variables known to correlate well to the dependent variable and as set force in procedures 86 and 87 (it is a source of contentions debate as to whether correlation is the most useful means to select variables in non-linear modelling techniques). While neural networks are tremendous at processing a very large number of features, this is often at the expense of generalisation and as such, the bug, encourages more care and thought in creating a more appropriate neural network:

NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 4)



Run the line of script to console, being prepared to wait a little while:



Upon the console returning, the neural network has been trained. Understanding the structure and performance of the neural network is a rather more complex affair than other procedures (which fits with the overall experience of using the package).

Procedure 2: Plotting a Neural Network.

It is often stated that a neural network is an unexplainable modelling techniques, which practically holds some truth, but to those with a background in regression modelling, explaining the model is not insurmountable.

The neuralnet object that was created in procedure 126, allows for the plotting of the neural network using the base plot() function. Simply call plot() passing the neural network object as an argument:

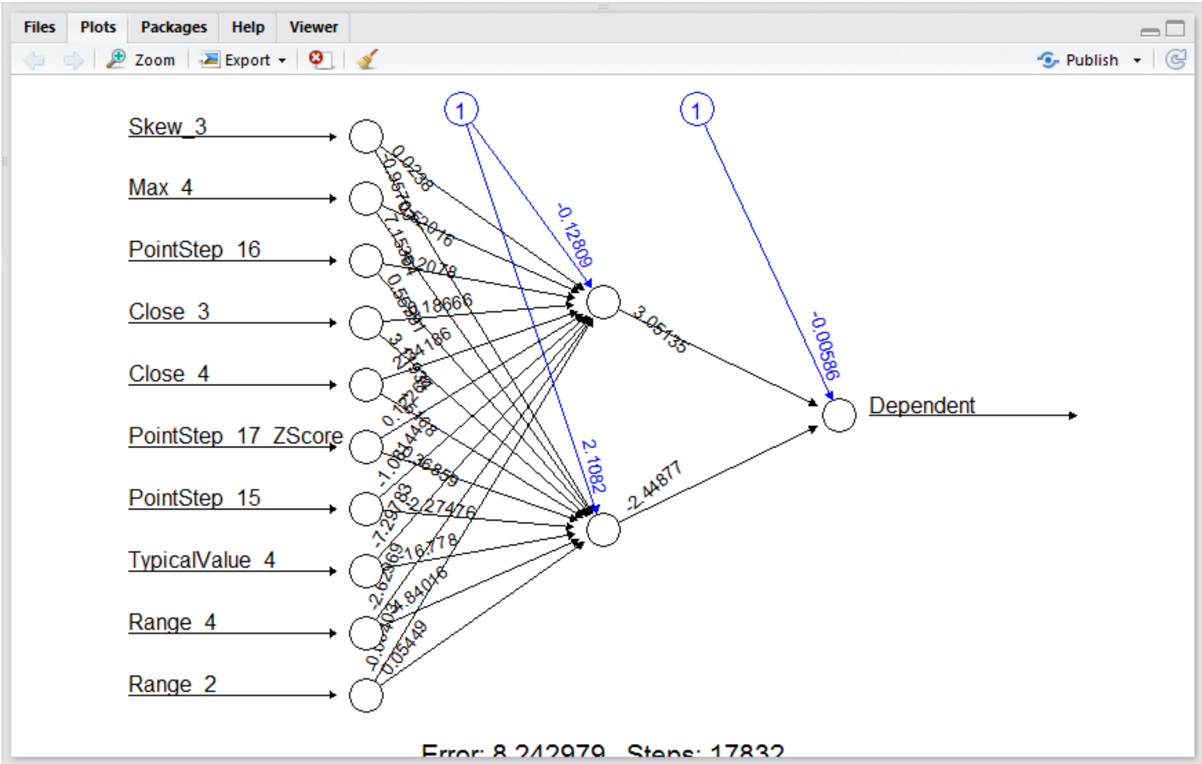
plot(NeuralNetworkFourByOne)

```r
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
```

Run the line of script to console:

```
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> summary(NeuralNetworkFourByOne)
                    Length Class      Mode
call                    4  -none-     call
response             2149  -none-     numeric
covariate           21490  -none-     numeric
model.list              2  -none-     list
err.fct                 1  -none-     function
act.fct                 1  -none-     function
linear.output           1  -none-     logical
data                  202  data.frame list
net.result              1  -none-     list
weights                 1  -none-     list
startweights            1  -none-     list
generalized.weights     1  -none-     list
result.matrix          28  -none-     numeric
> plot(NeuralNetworkFourByOne)
>
```

A plot is created of the neural network bearing stark resemblence to conceptual models put forward in this training manual, and in a model of less complexity, is in fact explainable and quite reproducable on a manual basis:



As the model becomes more and more complex, with the addition of more and more features, layers and processing elements, the neural network will naturally become less and less explaiable.

Procedure 3: Recalling a Neural Network with compute() and understanding performance.

The topology plot gives a useful window into the neural network, and its similarity to a regression model is unmistakable, there is none on the performance statistics associated with a regression model.

As this is a numeric prediction model, and not a classification model (although this is covered in procedure 130 as follows), we will use correlation to determine the relationship between the dependent variable and the predicted variable.

The compute() function is used instead of the predict() function, which returns an object with a few other properties rather than just the prediction (which would be easier). Something else to bear in mind is that the recall function, and indeed the training function, is very unforgiving in the event that the dependent variable has been passed (throwing an error Error in neurons[[i]] %*% weights[[i]] : non-conformable arguments). Frustratingly, it is necessary to subset the dataframe to return all columns explicitly, excluding the dependent variable, before passing it to the compute function. To recall the computed model:

```
ComputedModel <-
compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4",
"PointStep_17_ZScore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
```

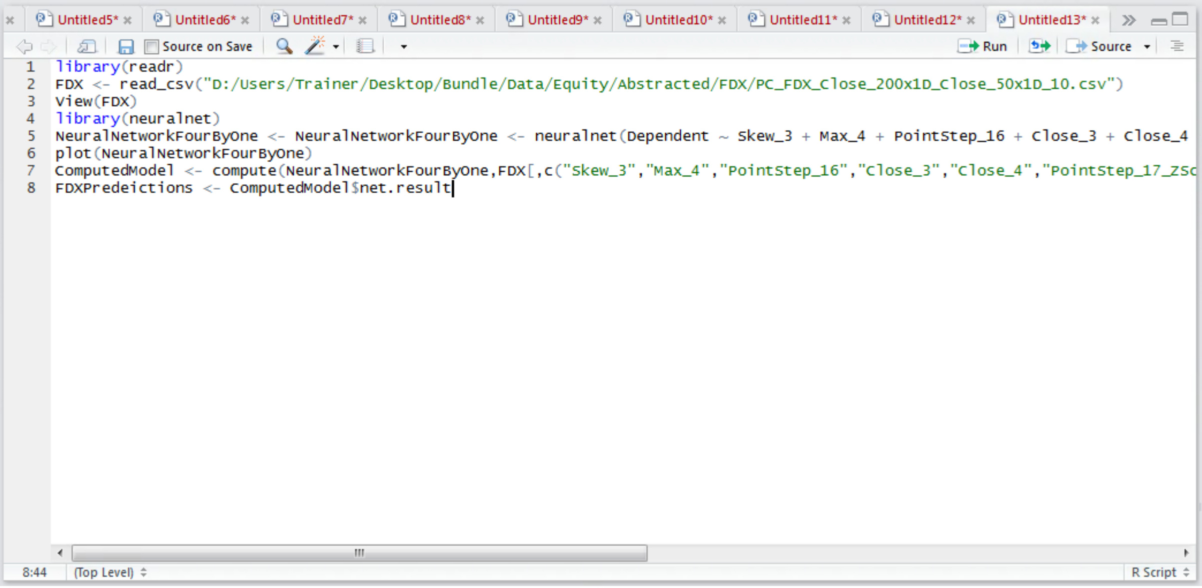
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
```

Run the line of script to console, it may take some time:

```
> library(readr)
> FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
Parsed with column specification:
cols(
  .default = col_double()
)
See spec(...) for full column specifications.
Warning: 2 parsing failures.
 row       col     expected                actual
2150 Dependent a double    (2149 row(s) affected)
2150 NA        202 columns 1 columns

> View(FDX)
> library(neuralnet)
Warning message:
package ‘neuralnet’ was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> |
```

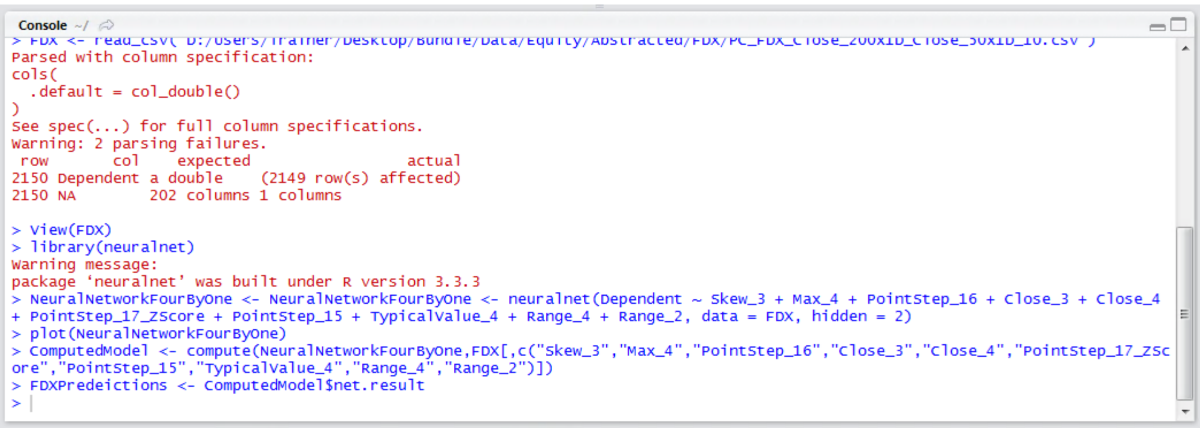
Unlike the predict method, compute has returned an object. It is necessary to extract the results from this object to a list, not a vector unfortunately, but that can be converted later using the unlist() function, using the net.result() method:

FDXPredeictions <- ComputedModel$net.result



```
1 library(readr)
2 FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
3 View(FDX)
4 library(neuralnet)
5 NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
6 plot(NeuralNetworkFourByOne)
7 ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
8 FDXPredeictions <- ComputedModel$net.result
```

Run the line of script to console:



```
> FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
Parsed with column specification:
cols(
  .default = col_double()
)
See spec(...) for full column specifications.
Warning: 2 parsing failures.
 row       col     expected             actual
2150 Dependent a double    (2149 row(s) affected)
2150 NA        202 columns 1 columns

> View(FDX)
> library(neuralnet)
Warning message:
package ‘neuralnet’ was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictions <- ComputedModel$net.result
>
```

To gain an assessment of the level of performance of the predictions vs the actuals, the correlation function can be used:

cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
```

Run the line of script to console:

```
  .default = col_double()
)
See spec(...) for full column specifications.
Warning: 2 parsing failures.
 row       col     expected                  actual
2150 Dependent a double    (2149 row(s) affected)
2150 NA          202 columns 1 columns

> View(FDX)
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictions <- ComputedModel$net.result
> cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.6502417224
>
```

It can be seen, in this example, that a correlation of 0.65 has been achieved. Referencing the initial correlation matrix calculated on the same dataset in procedure 93, it can be seen that this is an absolutely fantastic uplift in performance from the input correlations in isolation.

For completeness, the FDXPredictions vector, after converting it from a list, should be merged into the FDX data frame, however, using a more complex neural network, in this case taking more hidden layers, improvement will be sought in the subsequent procedure.

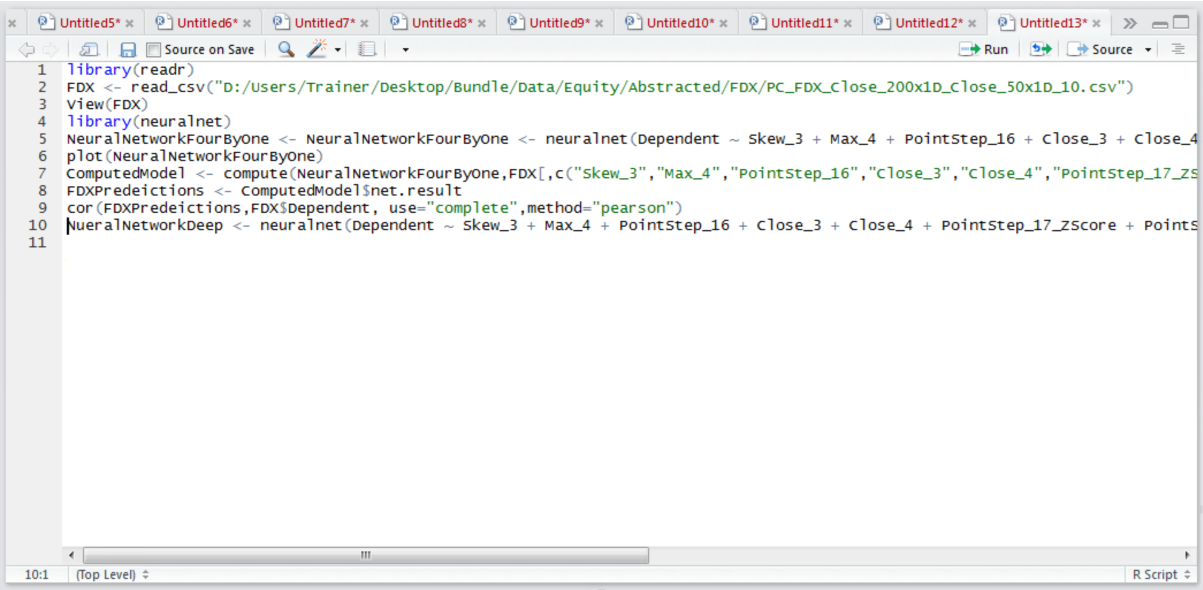
Procedure 4: Training a Deeper Neural Network.

In procedure 128, a neural network was trained having only a single hidden layer, albeit with several processing elements. Deep learning is the notion of having many more hidden layers and generally many more processing elements. Each layer is able to achieve abstraction autonomously, finding patterns that may not be apparent in manual abstraction. HOWEVER, it is lazy, adds valuable computational expense in recall (which begins to matter in super high throughput environments), as such deep learning can have circumvented to an extent, given more creativity in the abstraction phase.

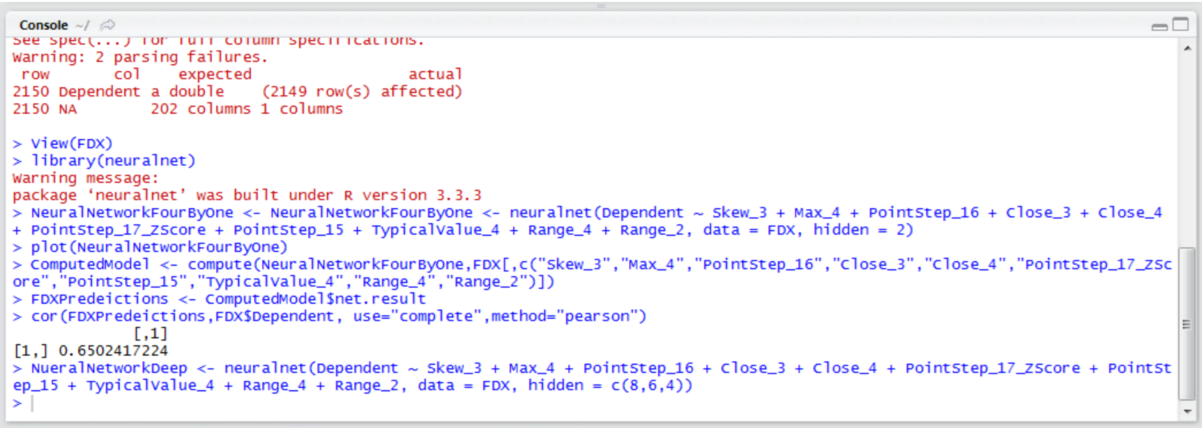
In this example, a much deeper neural network will be created where the same ten inputs will be used. The first hidden layer will have 8 processing elements, the second hidden layer will have 6 processing elements, the third hidden layer will have 4 processing elements yielding an output.

In procedure 126 a single value specifying just the number of processing elements was provided, where it was inferred that only a single hidden later is applicable. In this procedure, it is necessary to construct a vector, with each vector entry corresponding to a hidden layer, with the value of that hidden layer entry being the processing elements for that hidden layer:

NueralNetworkDeep <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))

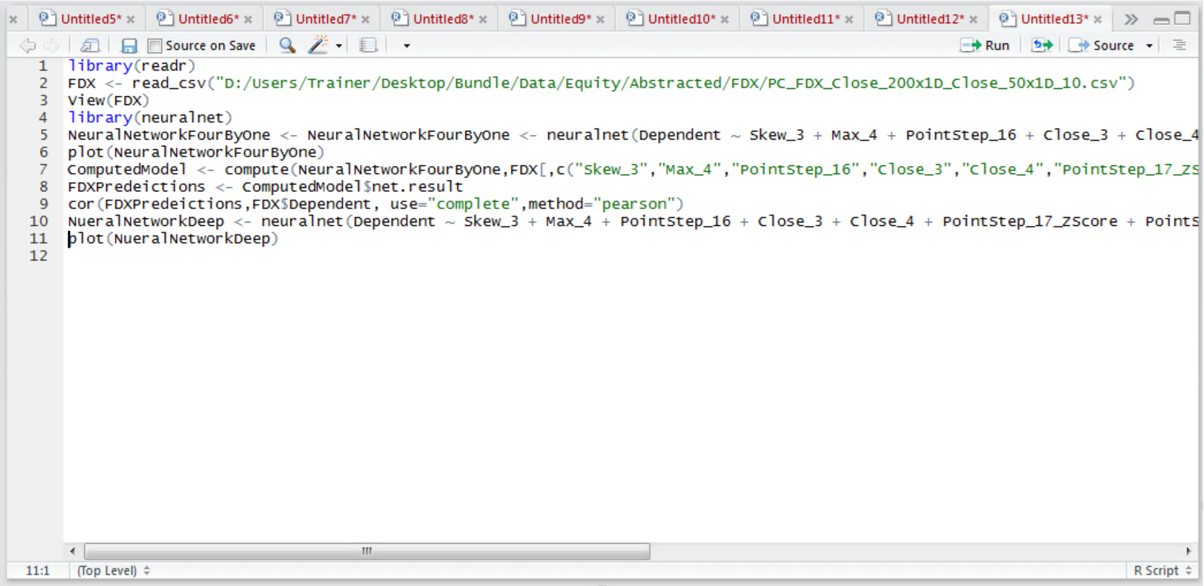


Run the line of script to console, expect it to take some time:

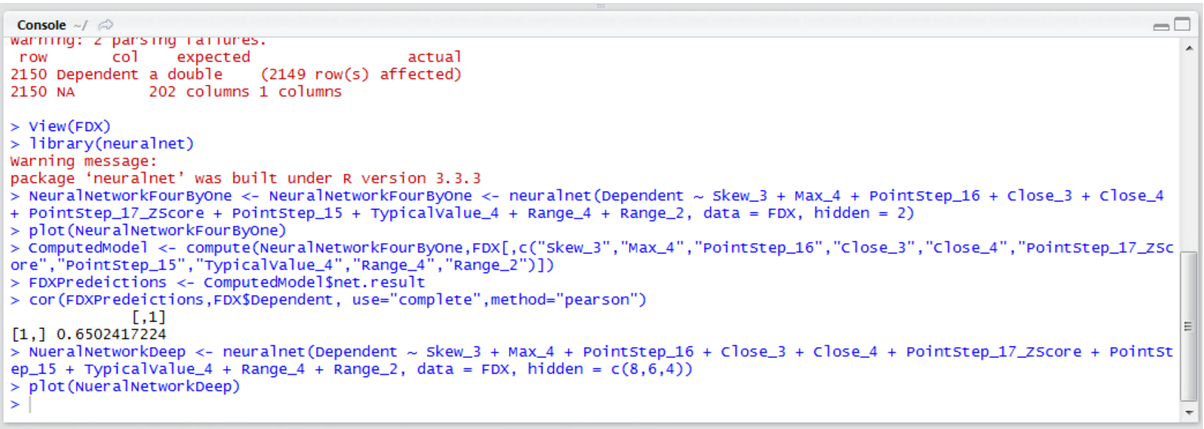


Plot the function to inspect the neural network:

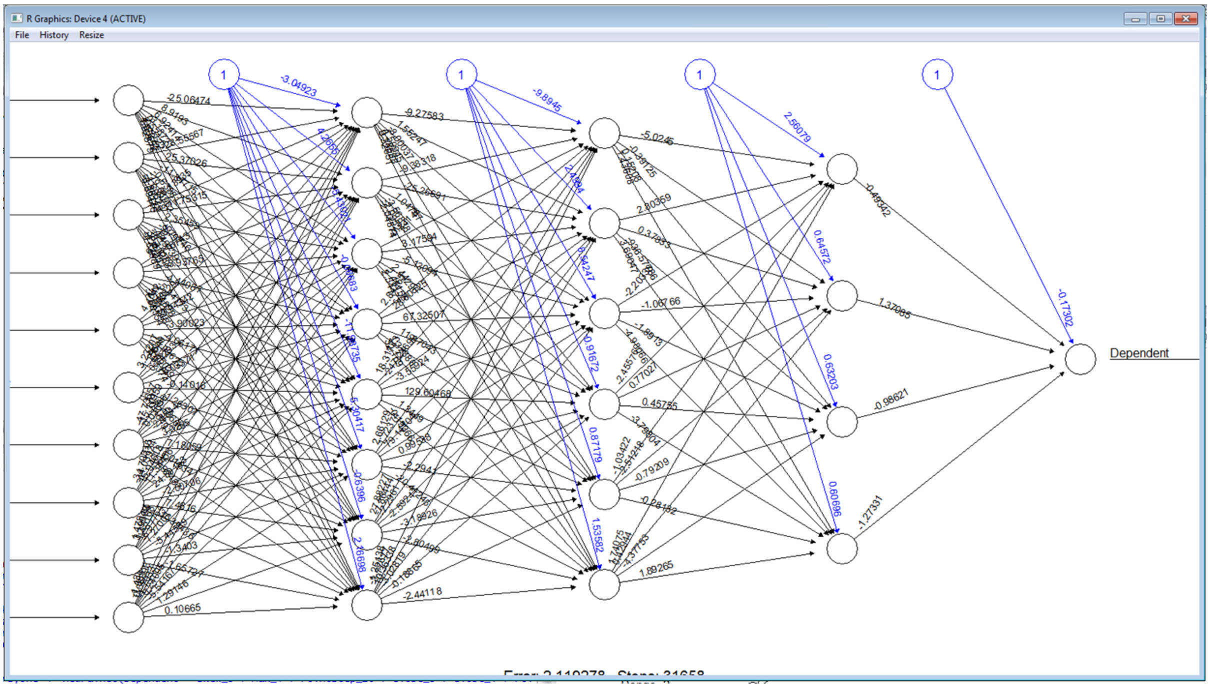
plot(NueralNetworkDeep)



Run the line of script to console:

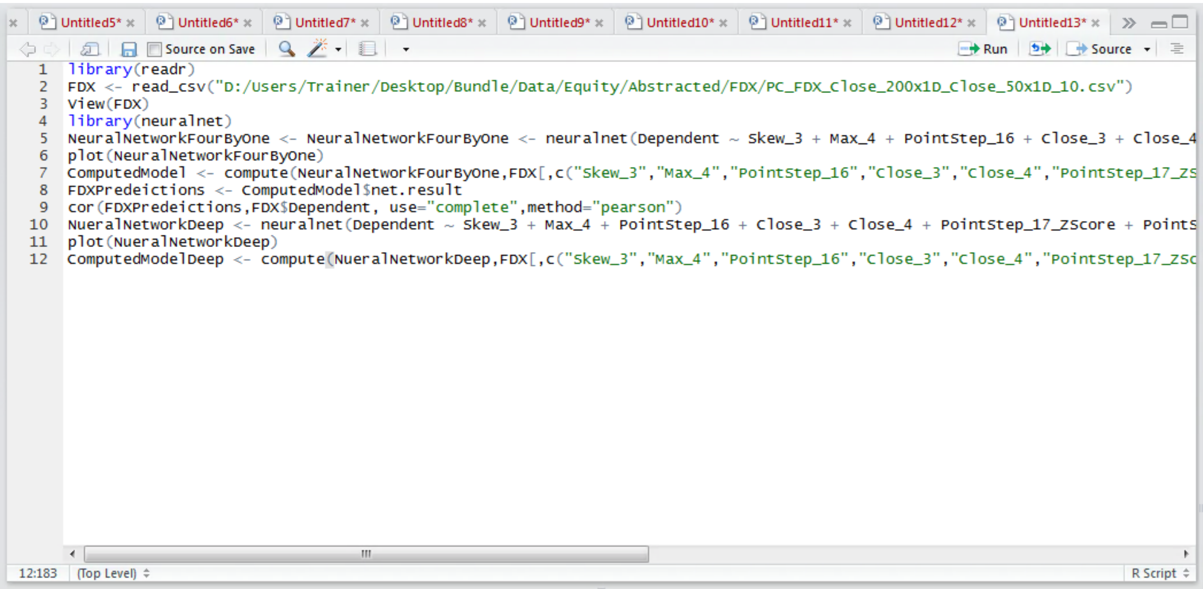


The plot has dramatically increased in complexity. It can be observed that the Neural Network now has three hidden layers, the first having 8 processing elements, the second having 6 processing elements and the third having 4 processing elements:



Naturally, this complexity is only worthwhile in the event that the classification accuracy has improved. As such, invoke compute and extract the results as per procedure 128:

ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZScore","PointStep_15","TypicalValue_4","Range_4","Range_2")])



Run the line of script to console:

```
Console ~/
2150 Dependent a double    (2149 row(s) affected)
2150 NA          202 columns 1 columns

> View(FDX)
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictions <- ComputedModel$net.result
> cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.6502417224
> NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointSt
ep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))
> plot(NueralNetworkDeep)
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSco
re","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> 
```

Extract the predictions to a list, for conversion to a vector later:

FDXPredeictionsDeep <- ComputedModelDeep$net.result

```
Untitled5* Untitled6* Untitled7* Untitled8* Untitled9* Untitled10* Untitled11* Untitled12* Untitled13*
Source on Save                                                        Run    Source
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
13:52   (Top Level)                                                               R Script
```

Run the line of script to console:

```
Console ~/
2150 NA          202 columns 1 columns

> View(FDX)
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictions <- ComputedModel$net.result
> cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.6502417224
> NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointSt
ep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))
> plot(NueralNetworkDeep)
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSco
re","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictionsDeep <- ComputedModelDeep$net.result
> 
```

Appraise the correlations between the predictions and the dependent variable:

cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
```

Run the line of script to console:

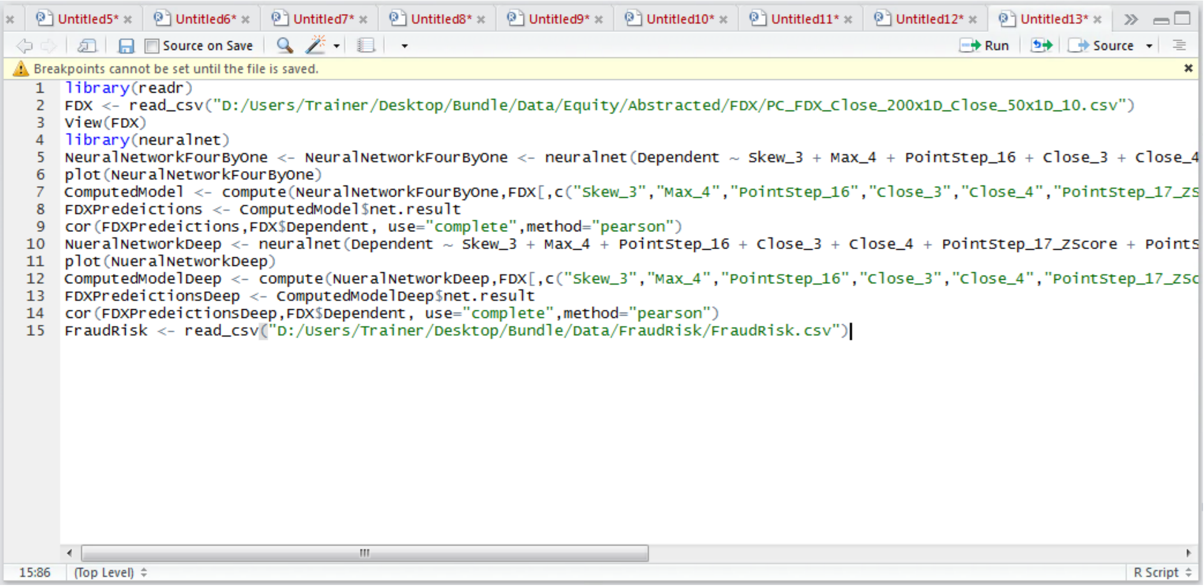
```
> library(neuralnet)
Warning message:
package ‘neuralnet’ was built under R version 3.3.3
> NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
+ PointStep_17_ZScore + PointStep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = 2)
> plot(NeuralNetworkFourByOne)
> ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
ore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictions <- ComputedModel$net.result
> cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.6502417224
> NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointSt
ep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))
> plot(NueralNetworkDeep)
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSco
re","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictionsDeep <- ComputedModelDeep$net.result
> cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.9184549566
>
```

It can be seen that the correlation between predicted and actual has leaped to a staggering 0.91 in response to increasing the complexity of the model.

Procedure 5: Training a Classification Model.

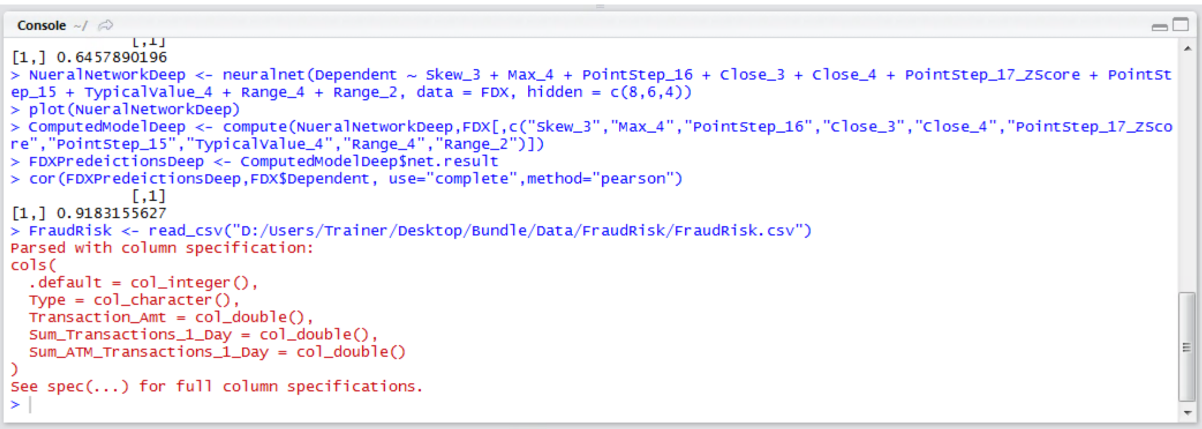
Neural Networks are universal classifiers, which means to say that they can be used as well on numeric prediction as classification. It won’t have escaped notice however that the internal weights comprising the neural network are all numeric coefficients. It follows that all input and output variables should be numeric also (via categorical data pivoting to 1 / 0, unfortunately not being able to rely on neuralnet() to interpret factors). In this example, a dataset of transactions where half of the transactions are fraud and half genuine, will be used as in procedure 98. Start by importing the FraudRisk dataset:

FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")



```
Untitled5* Untitled6* Untitled7* Untitled8* Untitled9* Untitled10* Untitled11* Untitled12* Untitled13*
Source on Save                                                    Run    Source
Breakpoints cannot be set until the file is saved.
1  library(readr)
2  FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
3  View(FDX)
4  library(neuralnet)
5  NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
6  plot(NeuralNetworkFourByOne)
7  ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
8  FDXPredeictions <- ComputedModel$net.result
9  cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
10 NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
11 plot(NueralNetworkDeep)
12 ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
13 FDXPredeictionsDeep <- ComputedModelDeep$net.result
14 cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
15 FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
15:86   (Top Level)                                                                R Script
```

Run the line of script to console:



```
Console ~/
      [,1]
[1,] 0.6457890196
> NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointSt
ep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))
> plot(NueralNetworkDeep)
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSco
re","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictionsDeep <- ComputedModelDeep$net.result
> cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
      [,1]
[1,] 0.9183155627
> FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
>
```

Once the FraudRisk data frame has been created, create a neural network of ten independent variables known to have strong correlation to the dependent variable with one hidden layer of four processing elements:

FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated + Has_Been_Abroad + Transaction_Amt + Different_Country_Transactions_1_Week + Different_Decline_Reasons_1_Day + Count_Transactions_Declined_1_Day + Count_In_Person_1_Day,data = FraudRisk, hidden = 4)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Transactions_1_Day + Authenticated + Count_Transactions_PIN_Decline_1
```

Run the line of script to console, it may take some time:

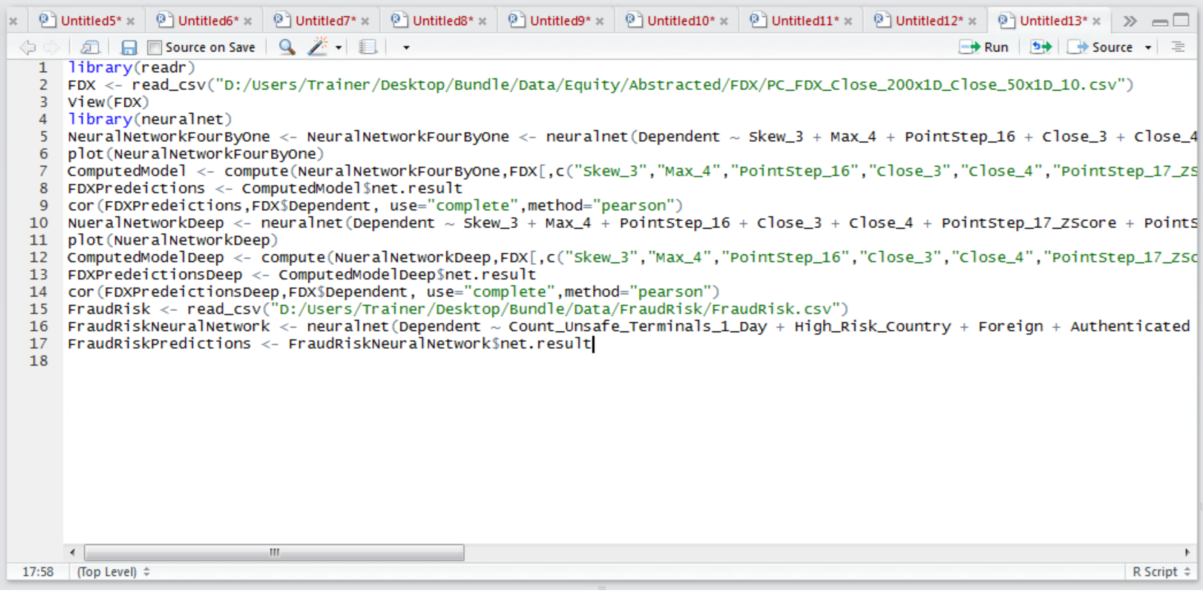
```
ep_15 + TypicalValue_4 + Range_4 + Range_2, data = FDX, hidden = c(8,6,4))
> plot(NueralNetworkDeep)
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSco
re","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictionsDeep <- ComputedModelDeep$net.result
> cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
             [,1]
[1,] 0.9090850292
> FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated +
 Has_Been_Abroad + Transaction_Amt + Different_Country_Transactions_1_Week + Different_Decline_Reasons_1_Day + Count_Transacti
ons_Declined_1_Day + Count_In_Person_1_Day,data = FraudRisk, hidden = 4)
> 
```

Once the console returns, the Neural Network has been trained upon the FraudRisk Dataset. For the purposes of this procedure it can be taken for granted that plot would return.

Procedure 6: Activating a Classification Model and Appraising Performance.

To recall the neural network, return a value between 0 and 1 depending on the likelihood that the record is fraudulent:

FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result



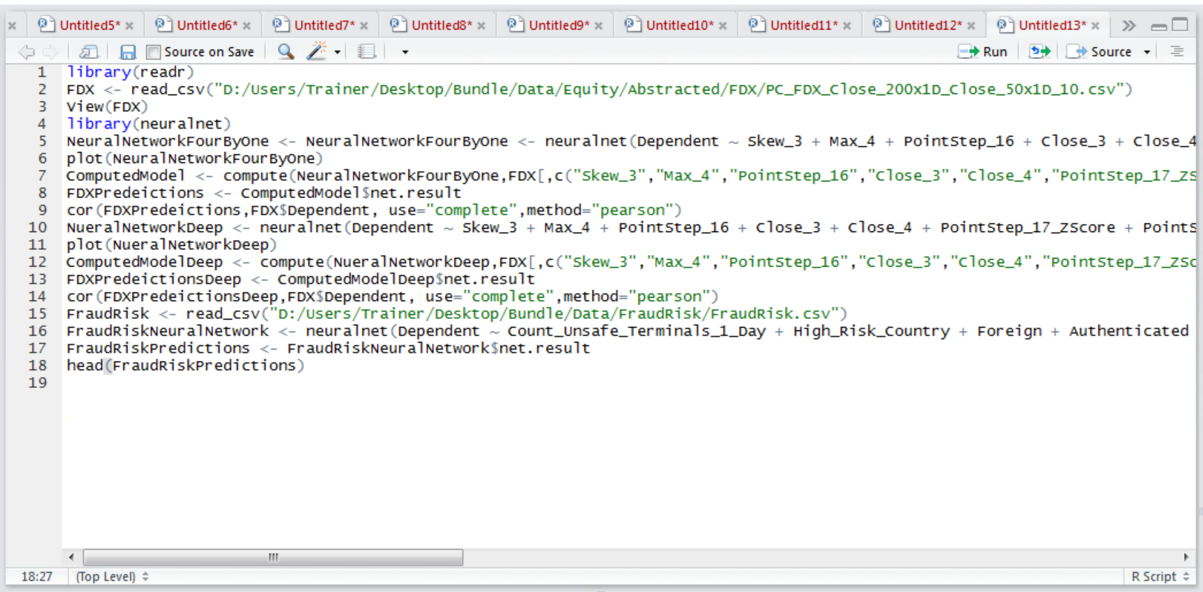
```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated
FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result
```

Run the line of script to console:

```
> ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZScore","PointStep_15","TypicalValue_4","Range_4","Range_2")])
> FDXPredeictionsDeep <- ComputedModelDeep$net.result
> cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
          [,1]
[1,] 0.9191813473
> FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated + Has_Been_Abroad + Transaction_Amt + Different_Country_Transactions_1_Week + Different_Decline_Reasons_1_Day + Count_Transactions_Declined_1_Day + Count_In_Person_1_Day,data = FraudRisk, hidden = 4)
> FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result
>
```

Peeking the results with the head() function:

head(FraudRiskPredictions)



```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated
FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result
head(FraudRiskPredictions)
```

Run the line of script to console:

```
Console ~/
982  0.86371626184
983  0.17452497436
984  0.84718762331
985  0.17632441455
986  0.17449192090
987  0.17448935004
988  0.17448827646
989  0.17448920140
990  0.17448717189
991  0.17820431355
992  0.17449598911
993  0.17448900255
994  0.17449907979
995  0.86335537847
996  0.86662731949
997  0.86345892557
998  0.90343642401
999  0.25103923982
1000 0.17448732790
 [ reached getOption("max.print") -- omitted 827 rows ]
>
```

It can be seen that numeric values, between 0 and 1, have been returned. The closer to one, the more likely that the record is fraudulent. To assert a proper classification, so that a confusion matrix may be plotted to appraise performance of the model, create a vector contains a 1 where the value of FraudRiskPredictions > 0.5, else 0, yet wrapping FraudRiskPrediction with the unlist() function to transform the list output to a vector:

IsFraud <- ifelse(unlist(FraudRiskPredictions) > 0.5,1,0)

```
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated
FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result
head(FraudRiskPredictions)
IsFraud <- ifelse(unlist(FraudRiskPredictions) > 0.5,1,0)
```

Run the line of script to console:

```
Console ~/
983  0.17452497436
984  0.84718762331
985  0.17632441455
986  0.17449192090
987  0.17448935004
988  0.17448827646
989  0.17448920140
990  0.17448717189
991  0.17820431355
992  0.17449598911
993  0.17448900255
994  0.17449907979
995  0.86335537847
996  0.86662731949
997  0.86345892557
998  0.90343642401
999  0.25103923982
1000 0.17448732790
 [ reached getOption("max.print") -- omitted 827 rows ]

> IsFraud <- ifelse(unlist(FraudRiskPredictions) > 0.5,1,0)
> 
```

As has become customary, use a confusion matrix to appraise the value of the classifier:

library("gmodels")

CrossTable(FraudRisk$Dependent, IsFraud)

```
Untitled5* Untitled6* Untitled7* Untitled8* Untitled9* Untitled10* Untitled11* Untitled12* Untitled13*
Source on Save                                                         Run   Source
library(readr)
FDX <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/Equity/Abstracted/FDX/PC_FDX_Close_200x1D_Close_50x1D_10.csv")
View(FDX)
library(neuralnet)
NeuralNetworkFourByOne <- NeuralNetworkFourByOne <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4
plot(NeuralNetworkFourByOne)
ComputedModel <- compute(NeuralNetworkFourByOne,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZS
FDXPredeictions <- ComputedModel$net.result
cor(FDXPredeictions,FDX$Dependent, use="complete",method="pearson")
NueralNetworkDeep <- neuralnet(Dependent ~ Skew_3 + Max_4 + PointStep_16 + Close_3 + Close_4 + PointStep_17_ZScore + PointS
plot(NueralNetworkDeep)
ComputedModelDeep <- compute(NueralNetworkDeep,FDX[,c("Skew_3","Max_4","PointStep_16","Close_3","Close_4","PointStep_17_ZSc
FDXPredeictionsDeep <- ComputedModelDeep$net.result
cor(FDXPredeictionsDeep,FDX$Dependent, use="complete",method="pearson")
FraudRisk <- read_csv("D:/Users/Trainer/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
FraudRiskNeuralNetwork <- neuralnet(Dependent ~ Count_Unsafe_Terminals_1_Day + High_Risk_Country + Foreign + Authenticated
FraudRiskPredictions <- FraudRiskNeuralNetwork$net.result
head(FraudRiskPredictions)
IsFraud <- ifelse(unlist(FraudRiskPredictions) > 0.5,1,0)
library("gmodels")
CrossTable(FraudRisk$Dependent, IsFraud)

21:41  (Top Level)                                                     R Script
```

Run the block of script to console:

```
Console ~/

                    | IsFraud 
FraudRisk$Dependent |         0 |         1 | Row Total | 
--------------------|-----------|-----------|-----------|
                  0 |       817 |       109 |       926 | 
                    |   191.425 |   230.448 |           | 
                    |     0.882 |     0.118 |     0.507 | 
                    |     0.819 |     0.131 |           | 
                    |     0.447 |     0.060 |           | 
--------------------|-----------|-----------|-----------|
                  1 |       181 |       720 |       901 | 
                    |   196.736 |   236.843 |           | 
                    |     0.201 |     0.799 |     0.493 | 
                    |     0.181 |     0.869 |           | 
                    |     0.099 |     0.394 |           | 
--------------------|-----------|-----------|-----------|
       Column Total |       998 |       829 |      1827 | 
                    |     0.546 |     0.454 |           | 
--------------------|-----------|-----------|-----------|

> 
```

In this example, it can be seen that 720 records were classified as being fraudulent correctly. In total, it can be seen that 901 records were classified, so the accuracy rate on predicting fraud is

79.9%, a substantial uplift on the logistic regression models created in procedure 93. It is well worth mentioning, that for classification problems, less is very often more and rather than increase network complexity by adding more and more hidden layers and processing elements, it is often more efficient to create many more abstracted variables backed by intuitive judgement and domain expertise.

Module 14: Exhaustive Search

Exhaustive is software that automates the search for Regression (Linear or Logistic) and Neural Networks Topology (Levenberg Marquart Learning). The software gains it name from the manner in which it will randomly trials topologies to arrive at an optimal, and tidy, model.

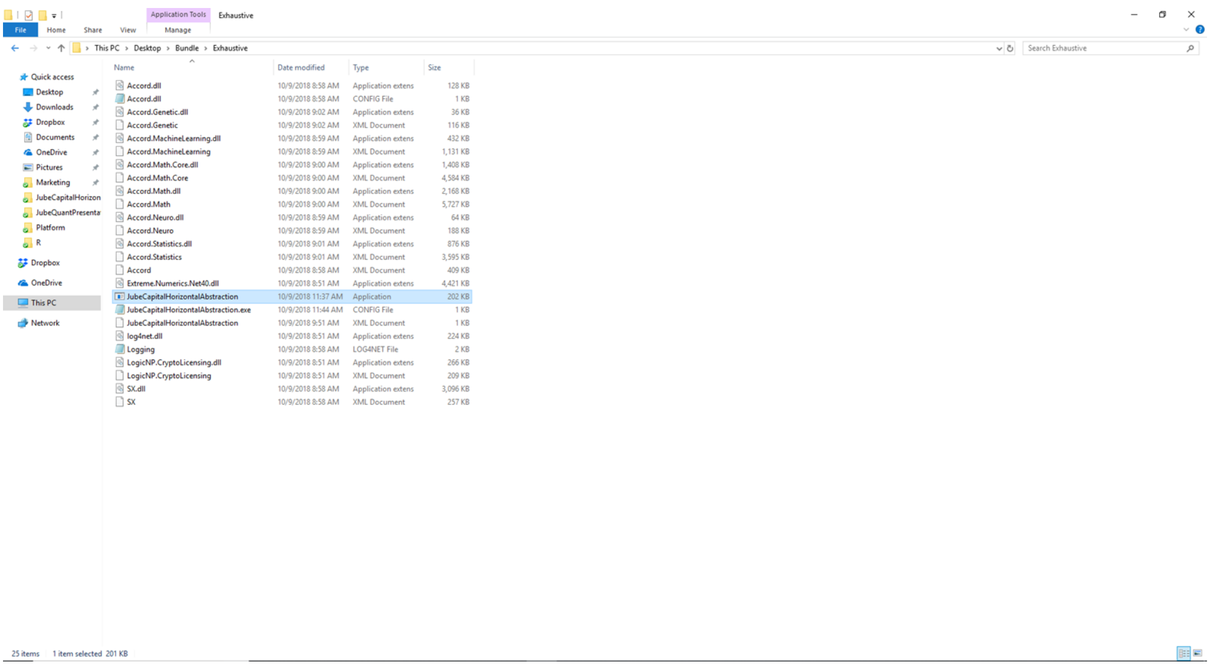
This module will focus on using Exhaustive for classification and will use the FraudRisk.csv AdTech.csv dataset.

These procedures assume that Exhaustive is already installed, however if this is not the case, the installation guide to install Exhaustive is available in the following location:

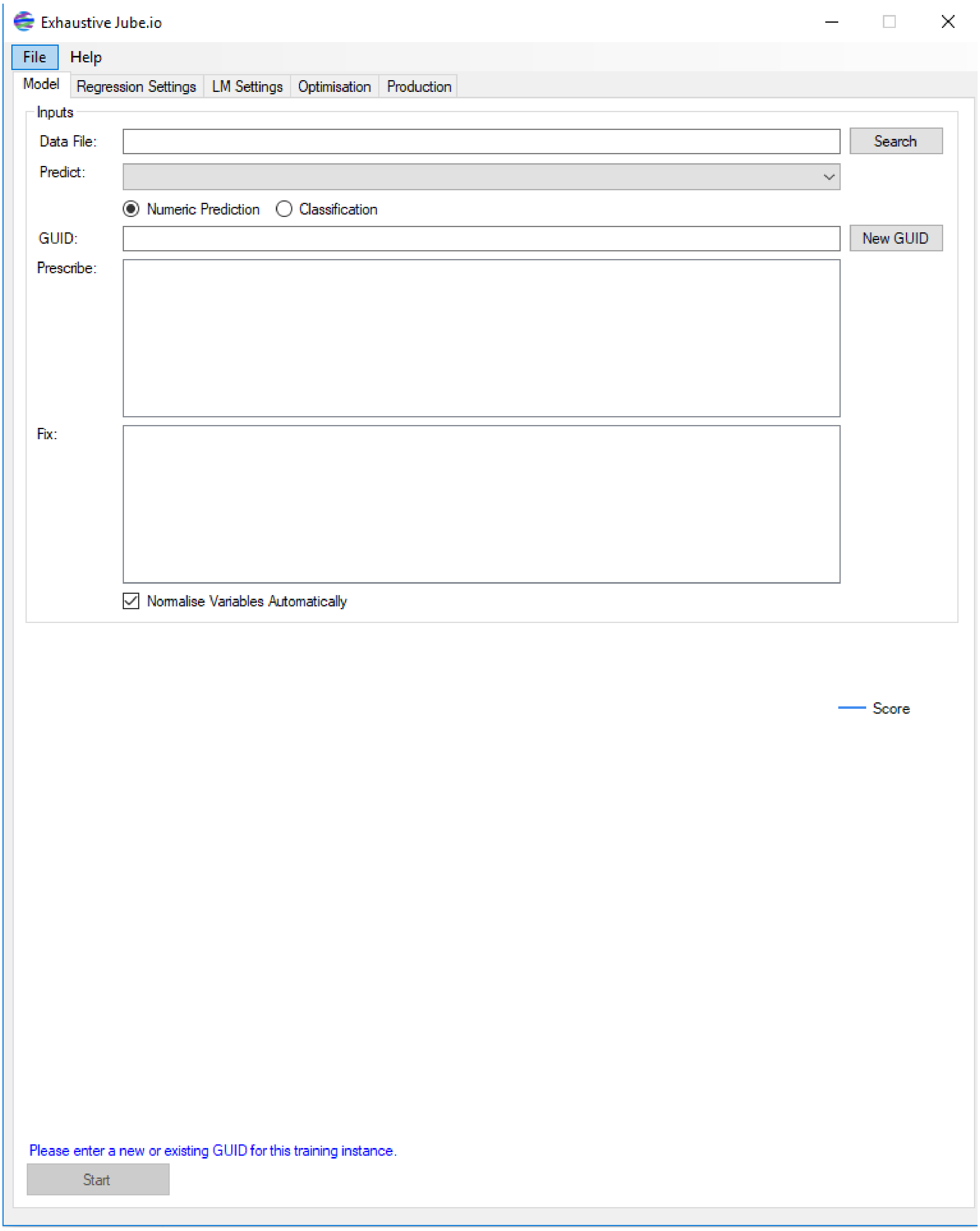
https://ui.jube.io/Help/Index.htm

Firstly, execute the Exhaustive program – which is a thick client application – by navigating to the directory:

Bundle\Exhaustive\



Execute the application titled JubeCapitalHorizontalAbstraction.exe:

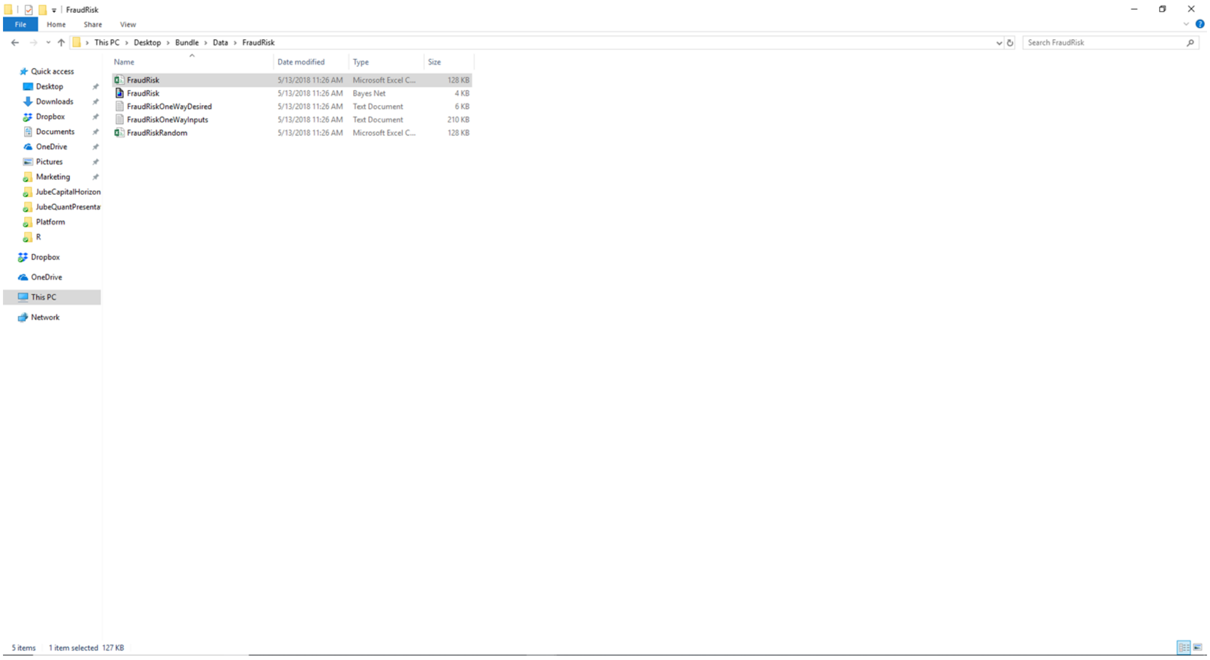


The Exhaustive thick client application will be loaded and available for use. The default parameters will be used throughout this training guide.

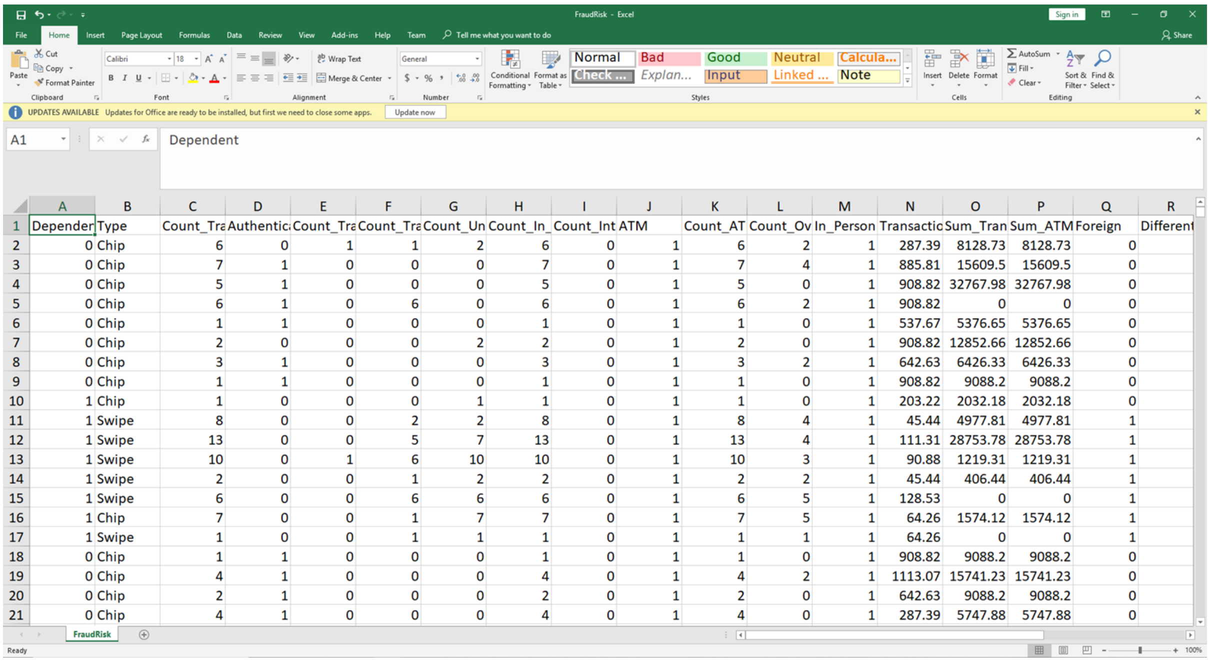
Procedure 1: Configure and Train a Classification Exhaustive Model

Once the Exhaustive application is loaded, the first step is to specify a csv file that is to be used for training. This file is typically structured such that the dependent variable is the very first column in the file, with the independent variables trailing that column. In this example, the FraudRisk.csv file will be used which is available as:

Bundle\Data\FraudRisk\FraudRisk.csv

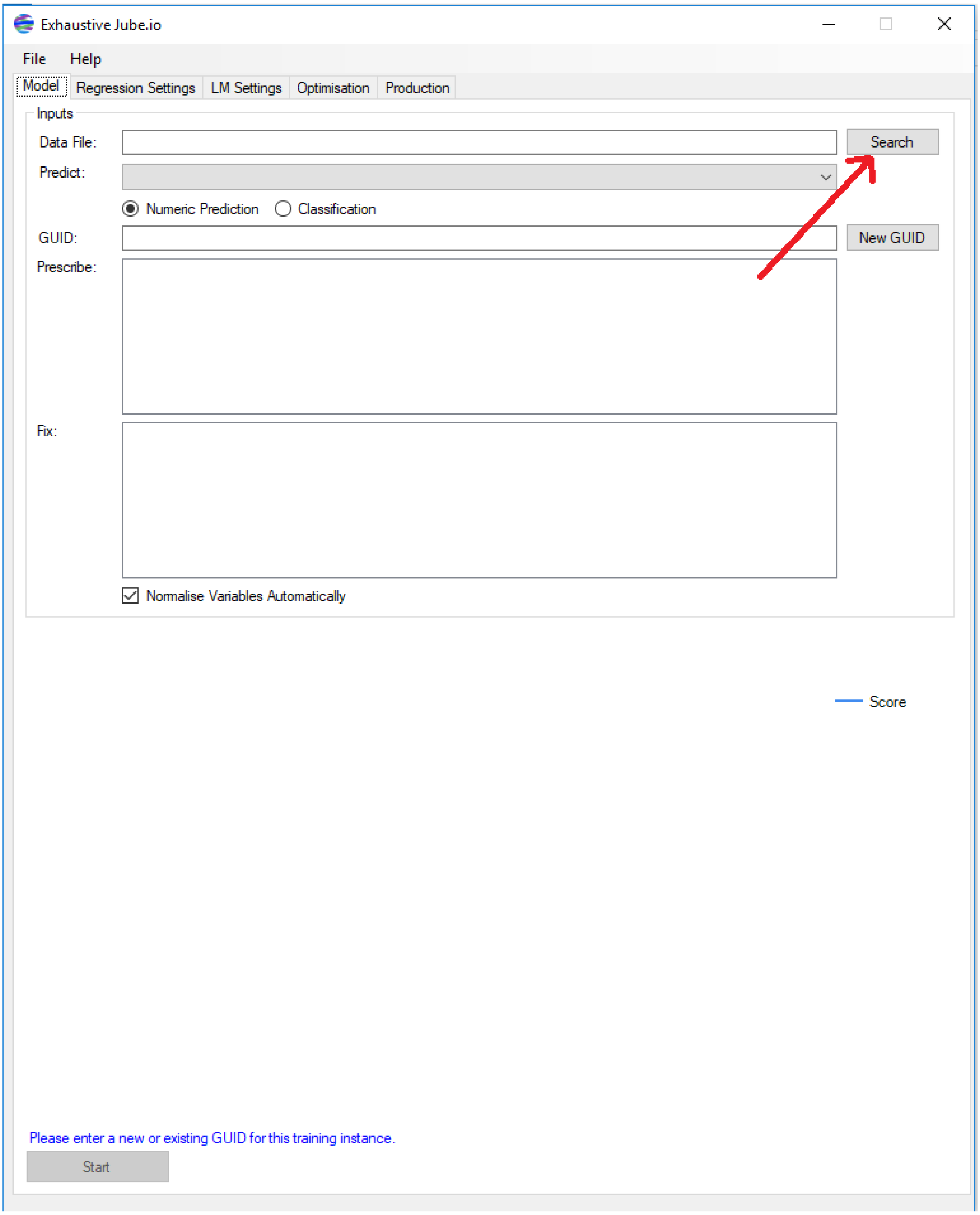


On inspection of this file in Excel it can be seen that the file is structured as aforementioned and as below:



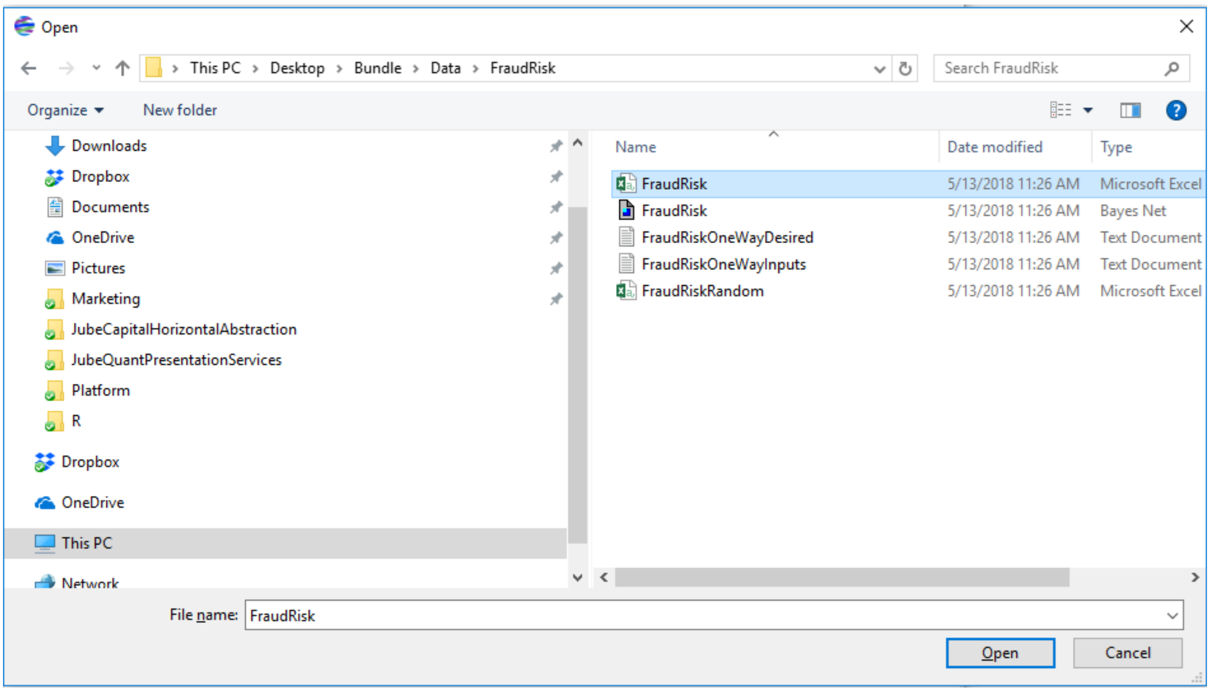
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Depender	Type	Count_Tra	Authentic	Count_Tra	Count_Tra	Count_Un	Count_In_	Count_Int	ATM	Count_AT	Count_Ov	In_Person	Transactic	Sum_Tran	Sum_ATM	Foreign	Different
2	0	Chip	6	0	1	1	2	6	0	1	6	2	1	287.39	8128.73	8128.73	0	
3	0	Chip	7	1	0	0	0	7	0	1	7	4	1	885.81	15609.5	15609.5	0	
4	0	Chip	5	1	0	0	0	5	0	1	5	0	1	908.82	32767.98	32767.98	0	
5	0	Chip	6	1	0	6	0	6	0	1	6	2	1	908.82	0	0	0	
6	0	Chip	1	1	0	0	0	1	0	1	1	0	1	537.67	5376.65	5376.65	0	
7	0	Chip	2	0	0	0	2	2	0	1	2	0	1	908.82	12852.66	12852.66	0	
8	0	Chip	3	1	0	0	0	3	0	1	3	2	1	642.63	6426.33	6426.33	0	
9	0	Chip	1	1	0	0	0	1	0	1	1	0	1	908.82	9088.2	9088.2	0	
10	1	Chip	1	0	0	0	1	1	0	1	1	0	1	203.22	2032.18	2032.18	0	
11	1	Swipe	8	0	0	2	2	8	0	1	8	4	1	45.44	4977.81	4977.81	1	
12	1	Swipe	13	0	0	5	7	13	0	1	13	4	1	111.31	28753.78	28753.78	1	
13	1	Swipe	10	0	1	6	10	10	0	1	10	3	1	90.88	1219.31	1219.31	1	
14	1	Swipe	2	0	0	1	2	2	0	1	2	2	1	45.44	406.44	406.44	1	
15	1	Swipe	6	0	0	6	6	6	0	1	6	5	1	128.53	0	0	1	
16	1	Chip	7	0	0	1	7	7	0	1	7	5	1	64.26	1574.12	1574.12	1	
17	1	Swipe	1	0	0	1	1	1	0	1	1	1	1	64.26	0	0	1	
18	0	Chip	1	1	0	0	0	1	0	1	1	0	1	908.82	9088.2	9088.2	0	
19	0	Chip	4	1	0	0	0	4	0	1	4	2	1	1113.07	15741.23	15741.23	0	
20	0	Chip	2	1	0	0	0	2	0	1	2	0	1	642.63	9088.2	9088.2	0	
21	0	Chip	4	1	0	0	0	4	0	1	4	0	1	287.39	5747.88	5747.88	0	

In the Exhaustive application, on the first tab titled Model and in the Inputs section, draw attention to the textbox titled Data File. This textbox is intended to accept the location of the csv file to be used in model training. The simplest means to complete the Data File textbox is to click on the Search button to expand the directory search tool:

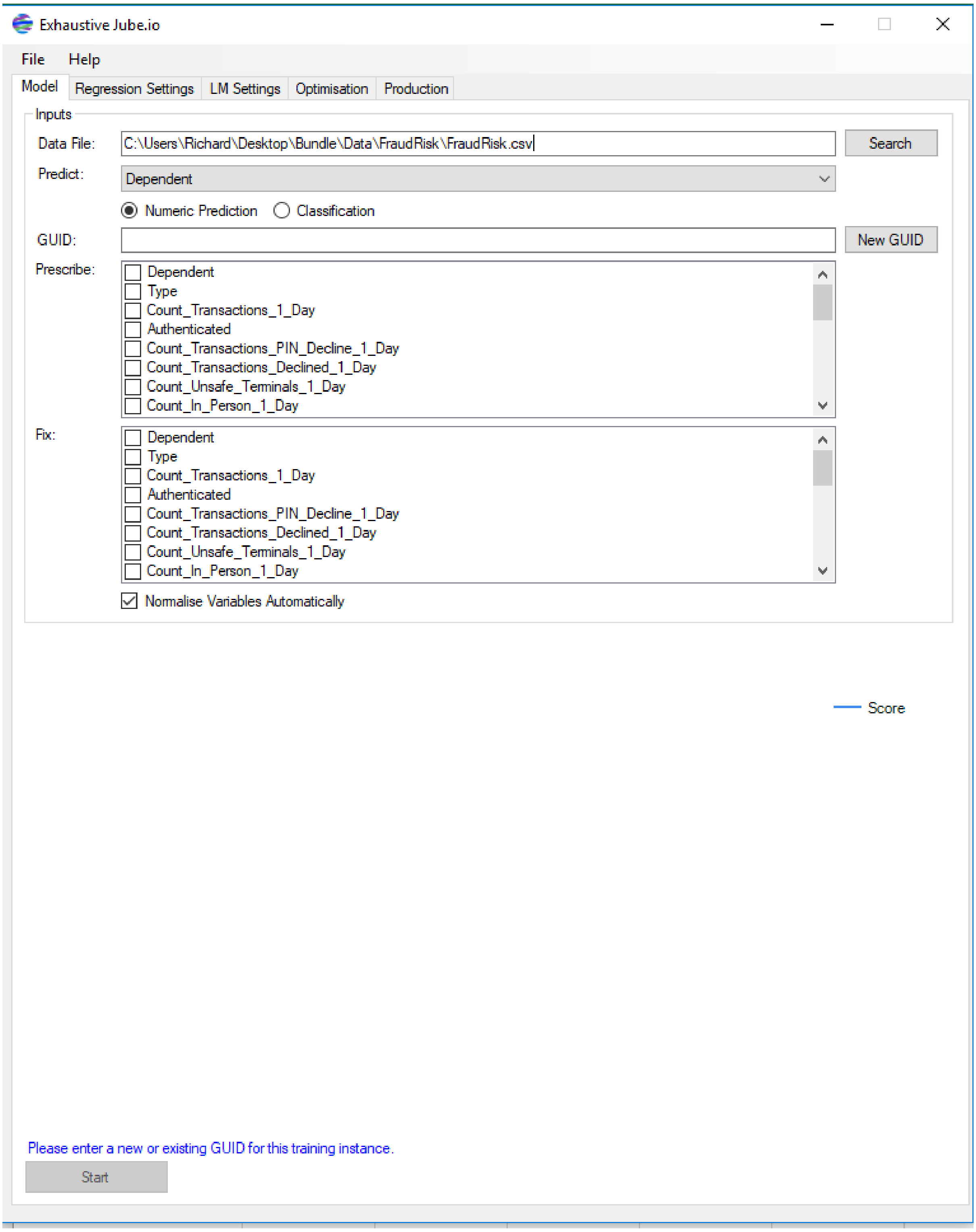


On clicking the Search button, the Directory and File browser will appear. Use this dialog box to navigate to the file FraudRisk.csv:

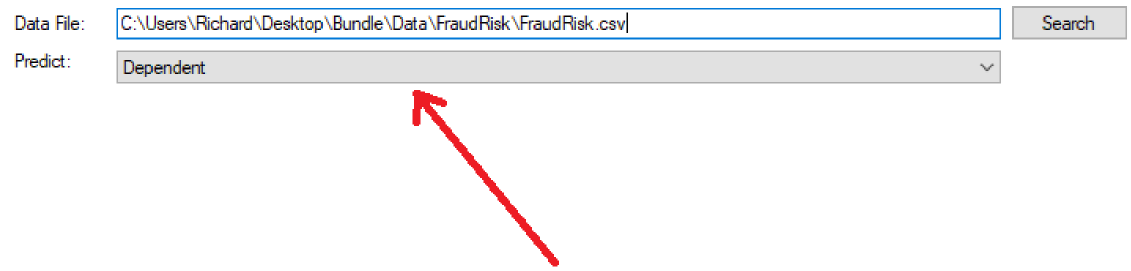
Bundle\Data\FraudRisk\FraudRisk.csv



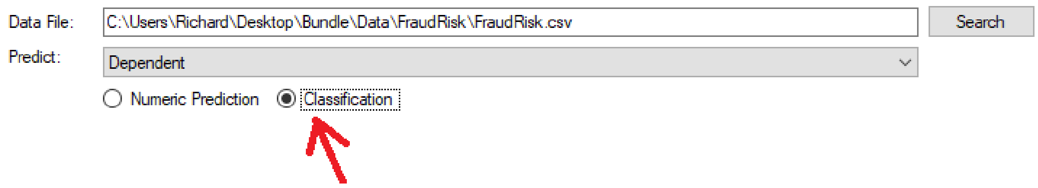
Upon navigating to the FraudRisk.csv file, click Open to place the file location in the Data File textbox:



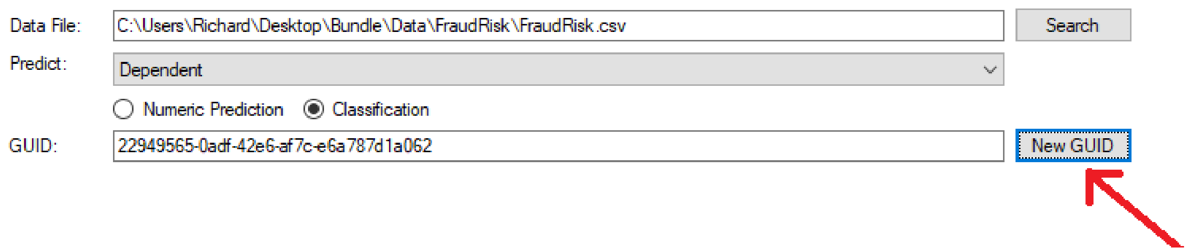
It can be seen that the File Headers have been used to populate several control boxes in the software. Drawing attention to the Predict drop down, set this value to the Dependent Variable, which in the case is titled Dependent:



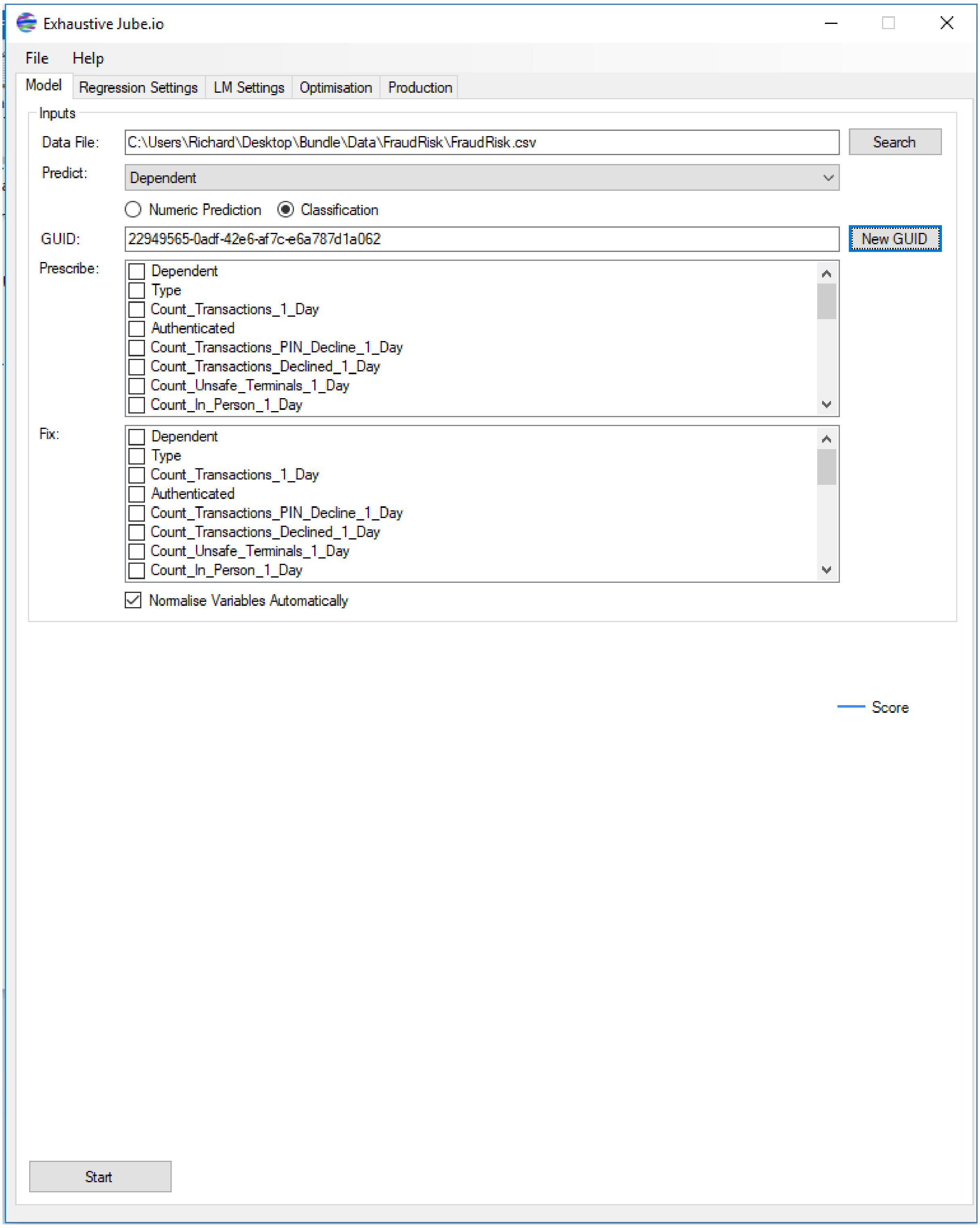
Fraud Risk is a classification problem, and as such, set the Classification radio button:



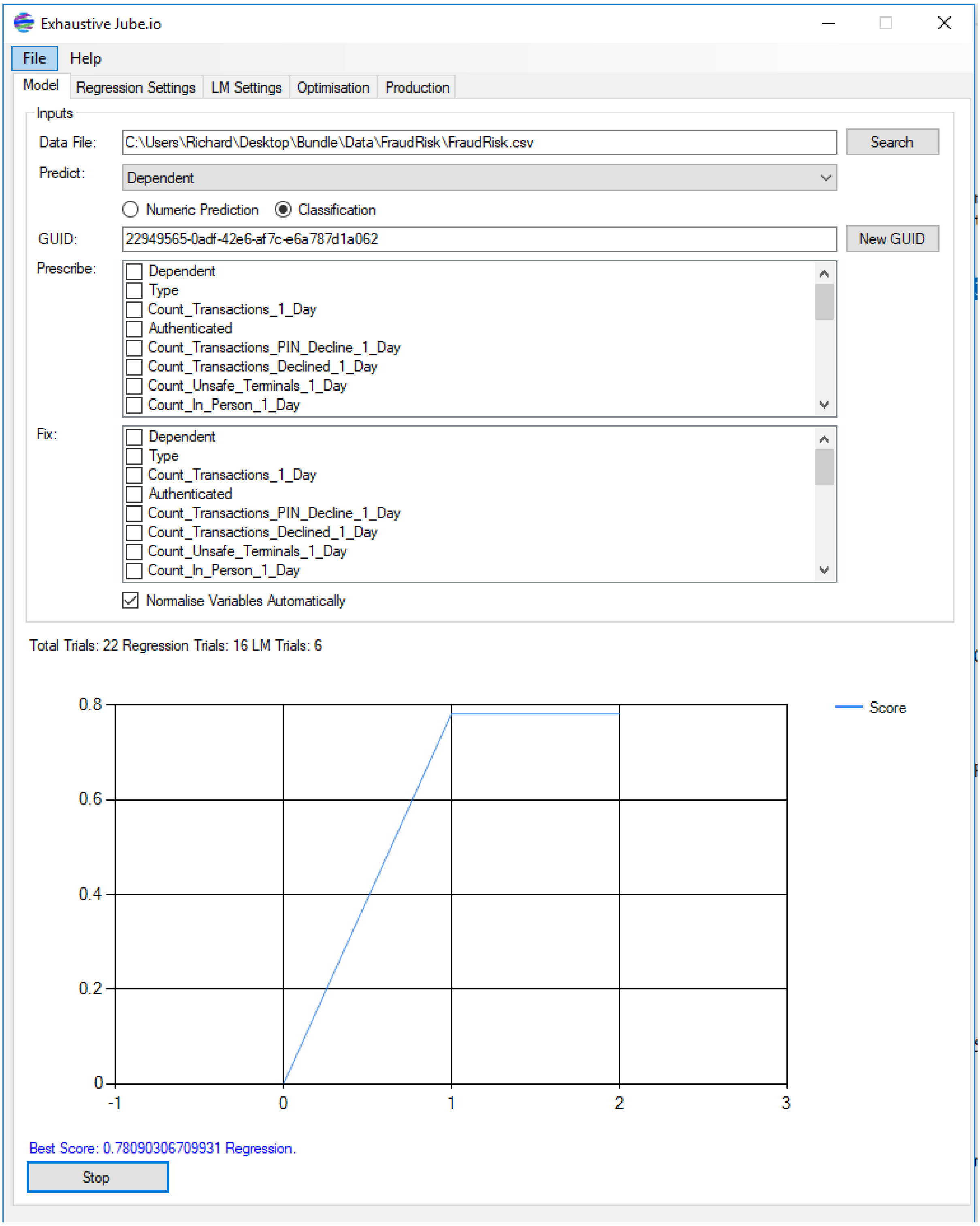
Exhaustive stores its training process in an SQL Server database under a training instance. The training instance is allocated a GUID (a guaranteed unique value). To create a GUID, click the New GUID button which will populate a fresh GUID in the GUID textbox:



For this classification problem, there are no prescription variables and no variables to fix. The model is now ready to start training:



To start model training, click the Start Button towards the base of the tab. The status bar towards the base of the tab will feedback the training progress, alongside line chart report detailing the best model score and number of models attempted:



The model will keep running ad infinitum, or until the maximum number of trials is exceeded as specified in the Settings tabs. In this example, the best score achieved is 78, which would indicate that the average between Correlation and Percentage Correct is 78.

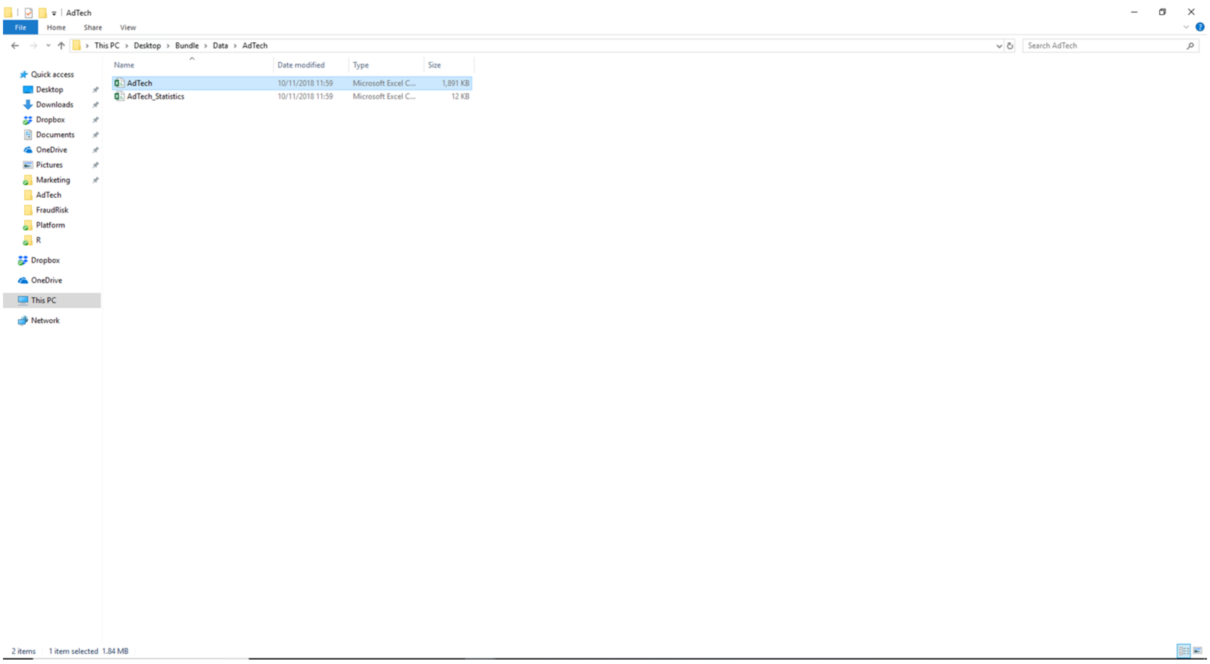
Procedure 2: Configure and Train a Prescriptive Exhaustive Model

One of the interesting and unique features in Exhaustive is the ability for models to be recalled where certain variables are randomised to observe the effect it has on the score at recall. Fluttering certain variables in this way can facilitate experimentation in real-time to prescribe an optimal solution to a problem.

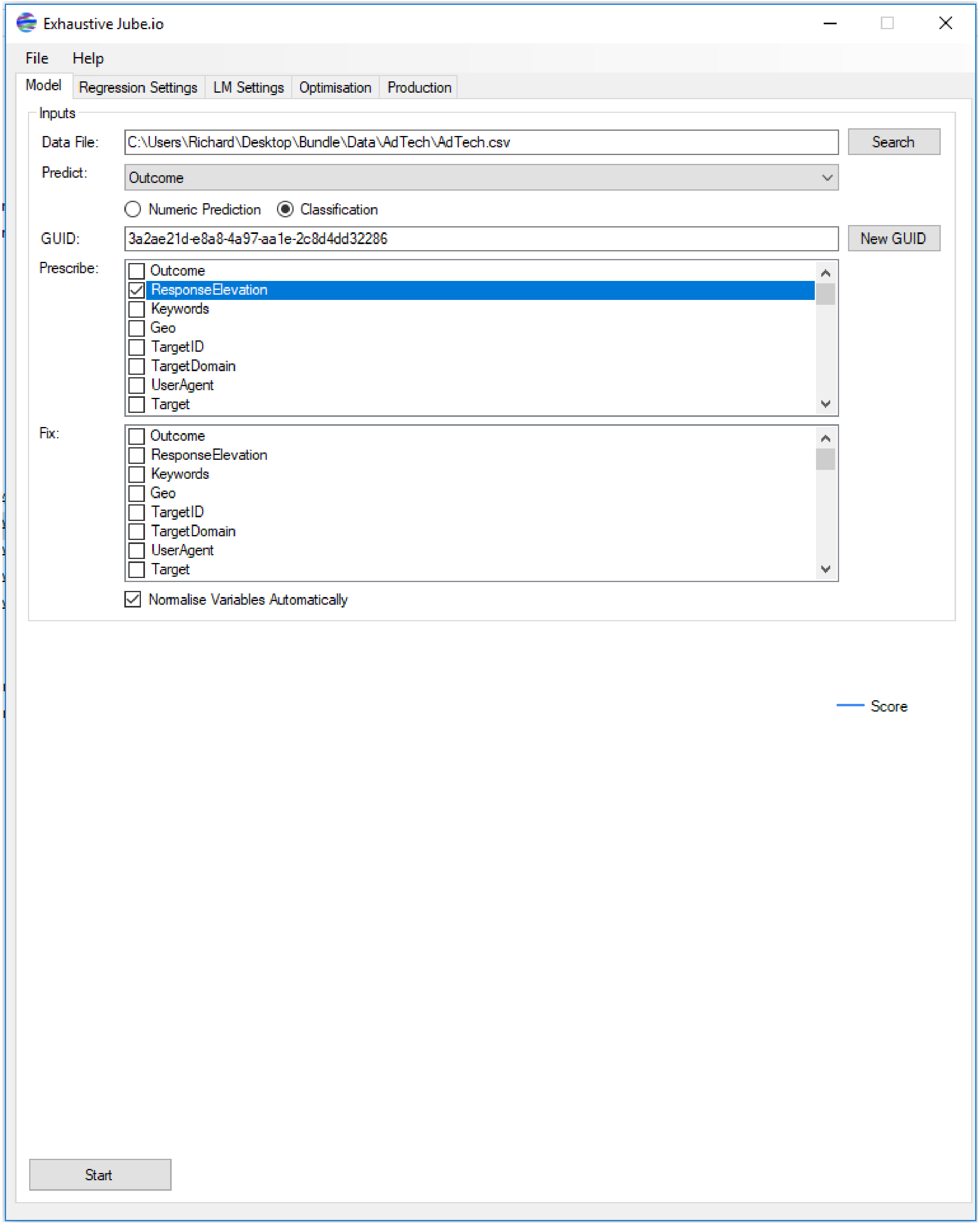
Creating a prescription model is exactly the same as creating other models in Exhaustive, with the additional step being the specification of variables that are to be used as prescription variables.

In this procedure, repeat the steps as detailed in procedure x, with the following file but stop short at clicking the Start button:

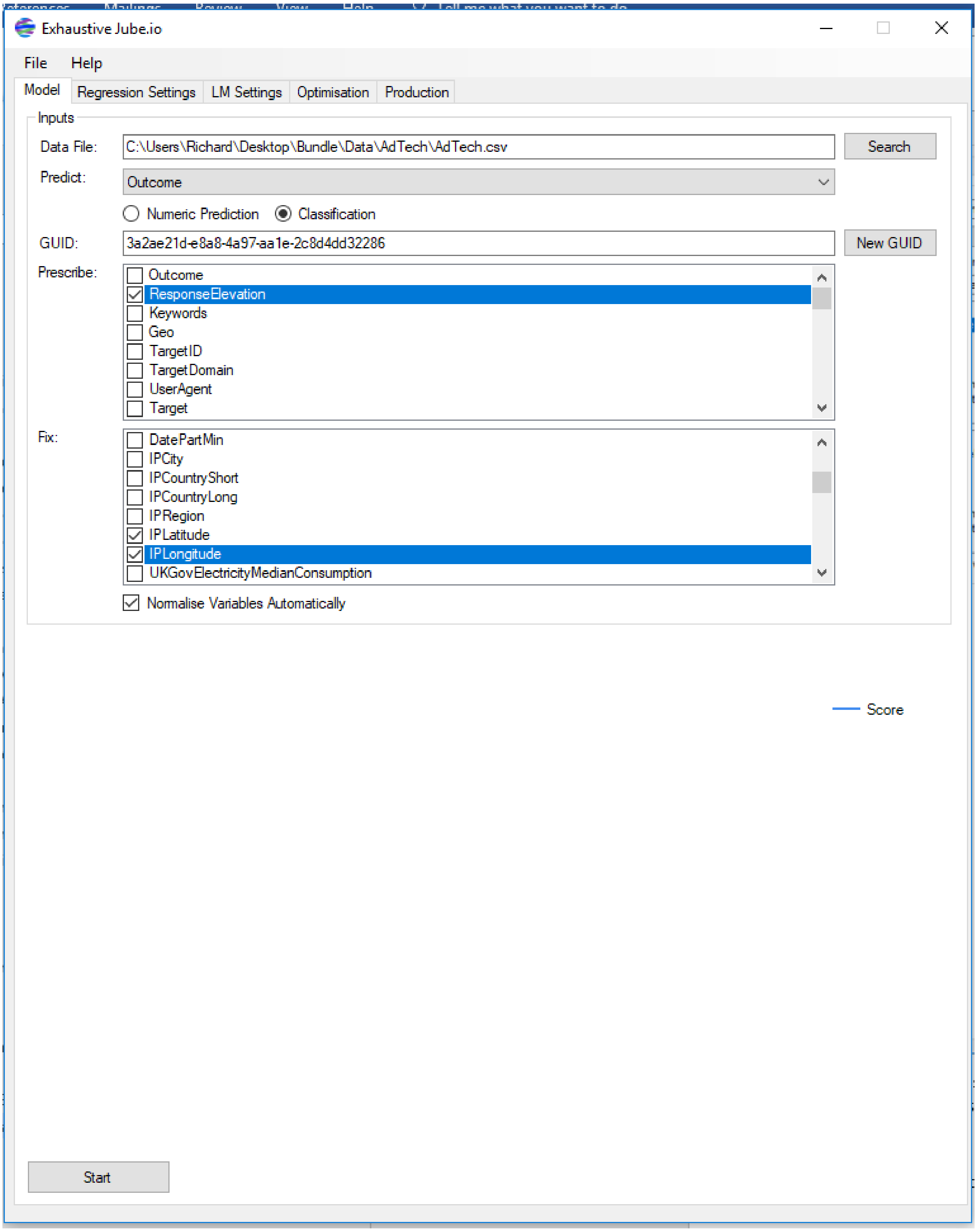
\Bundle\Data\AdTech\AdTech.csv



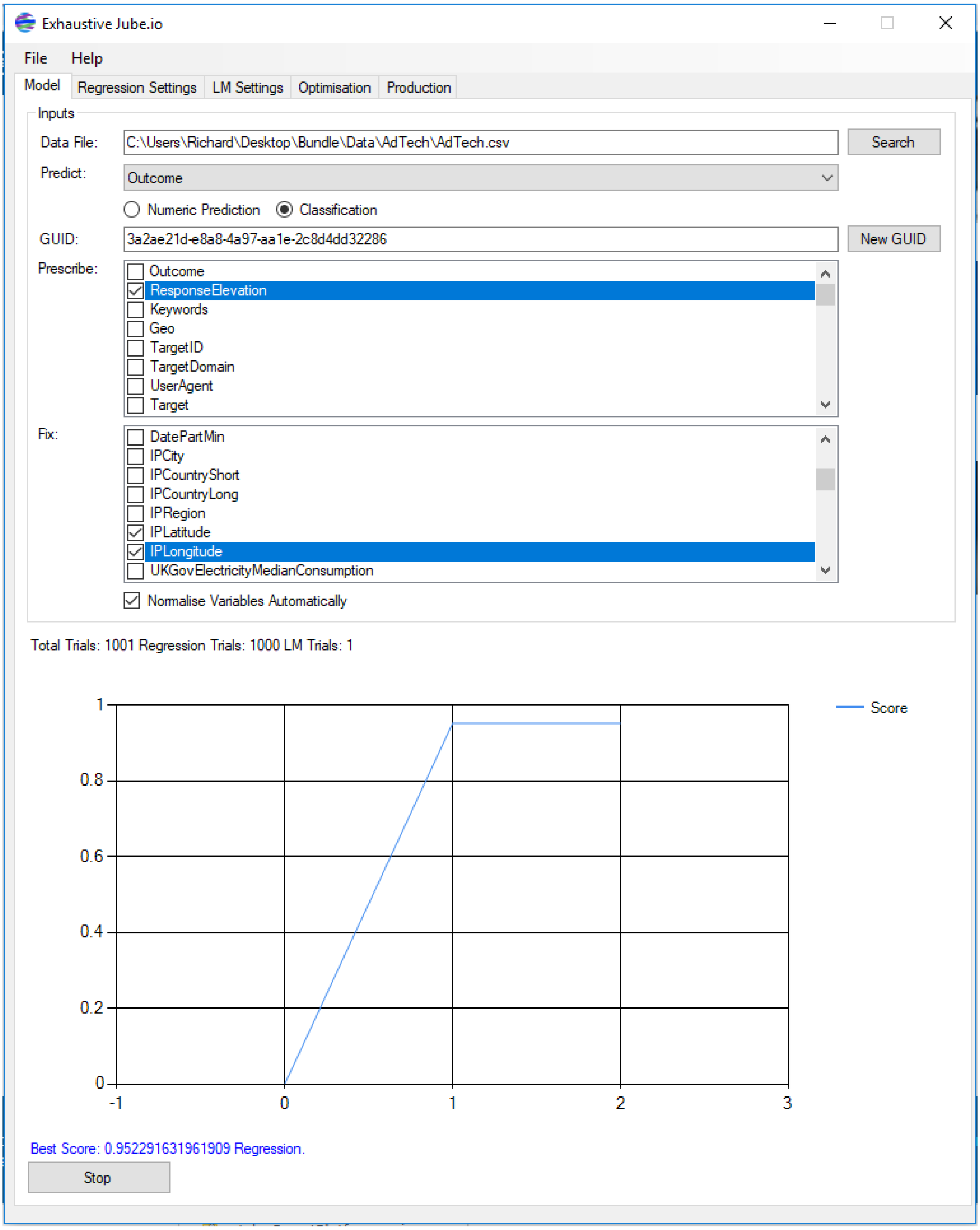
This is structure in the same manner as the FraudRisk.csv file, although there is a field called Response Elevation (i.e. bid) for which optimisation is sought. Specifying the variable as being Prescriptive instructs exhaustive to simulate the variable on model recall, rather than rely on what has been passed (if indeed such a value exists at the time of recall):



In this example, as it is thought that geography plays an important part in AdTech, fix the Latitude and Longitude fields such that these variables will be in an Exhaustive trial as a minimum:



Click on the Start button to begin the training as in procedure x:



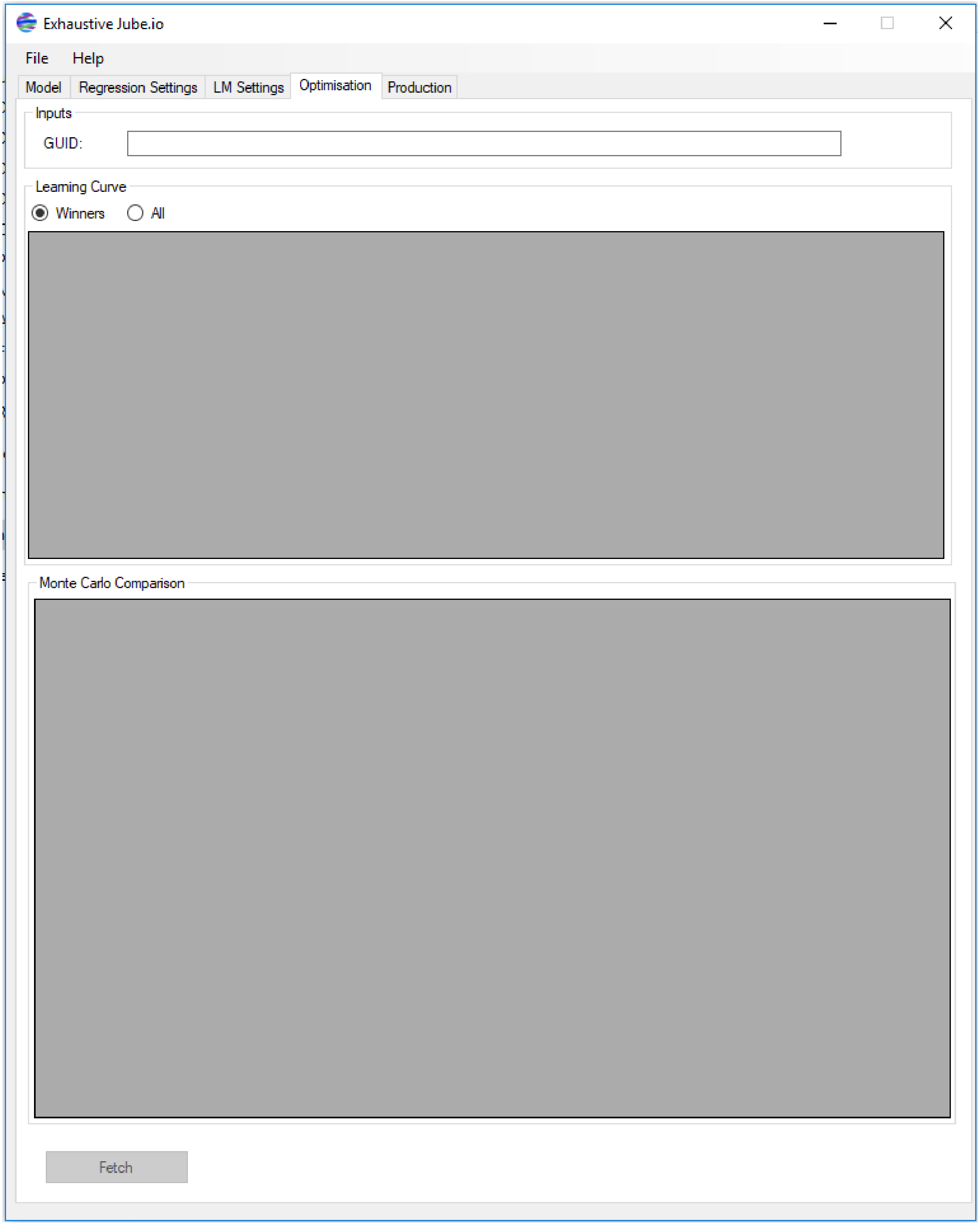
Procedure 3: Recall an Exhaustive Model

It has been observed that the a GUID is specified at the point the model is trained. This GUID is used to produce reports on the training process as well as facilitate model recall via batch file or API.

The GUID that will be used for this example is as follows, being the FraudRisk.csv model training outcome:

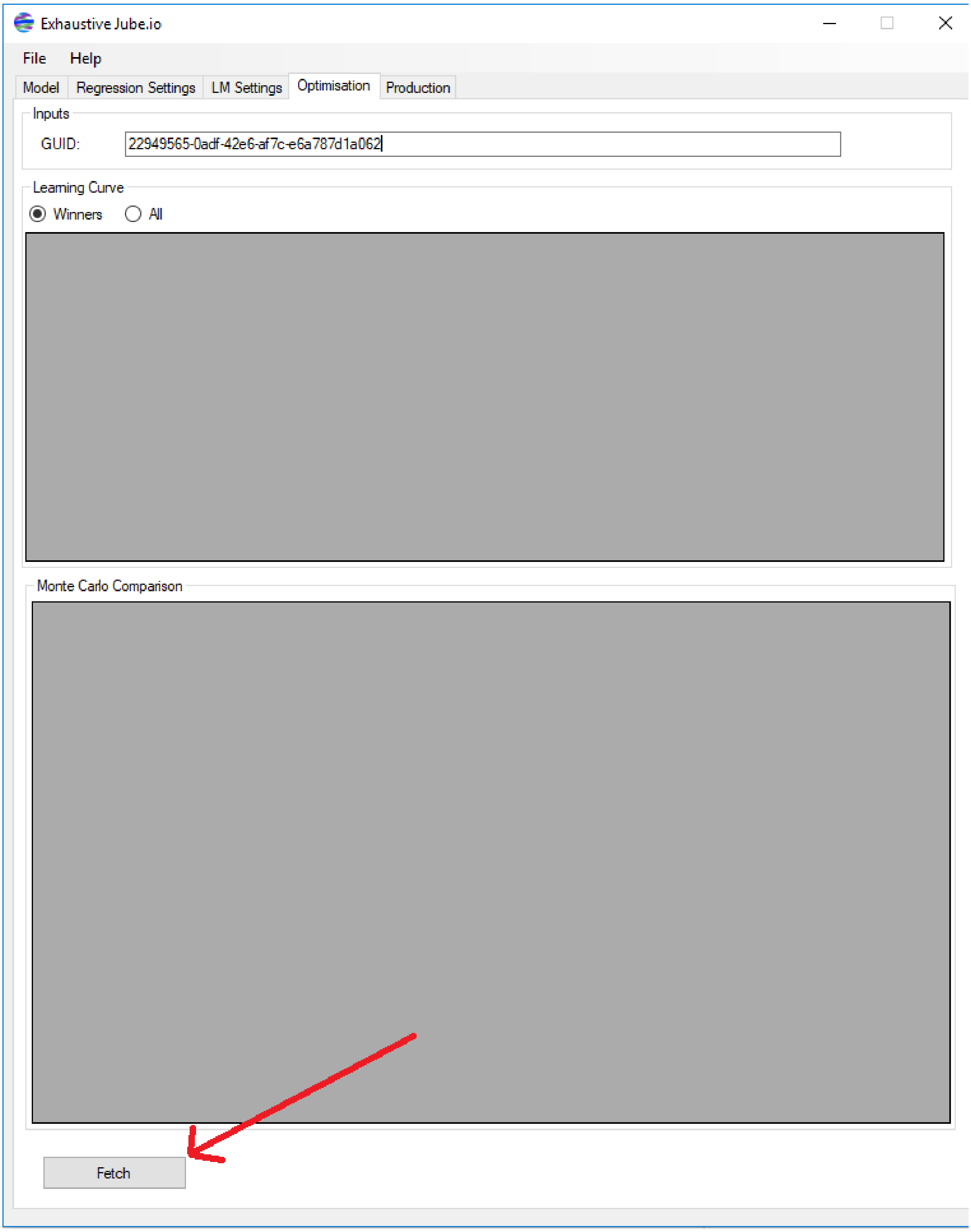
22949565-0adf-42e6-af7c-e6a787d1a062

To view the winning model for this GUID, start by clicking on the Optimisation tab in Exhaustive:

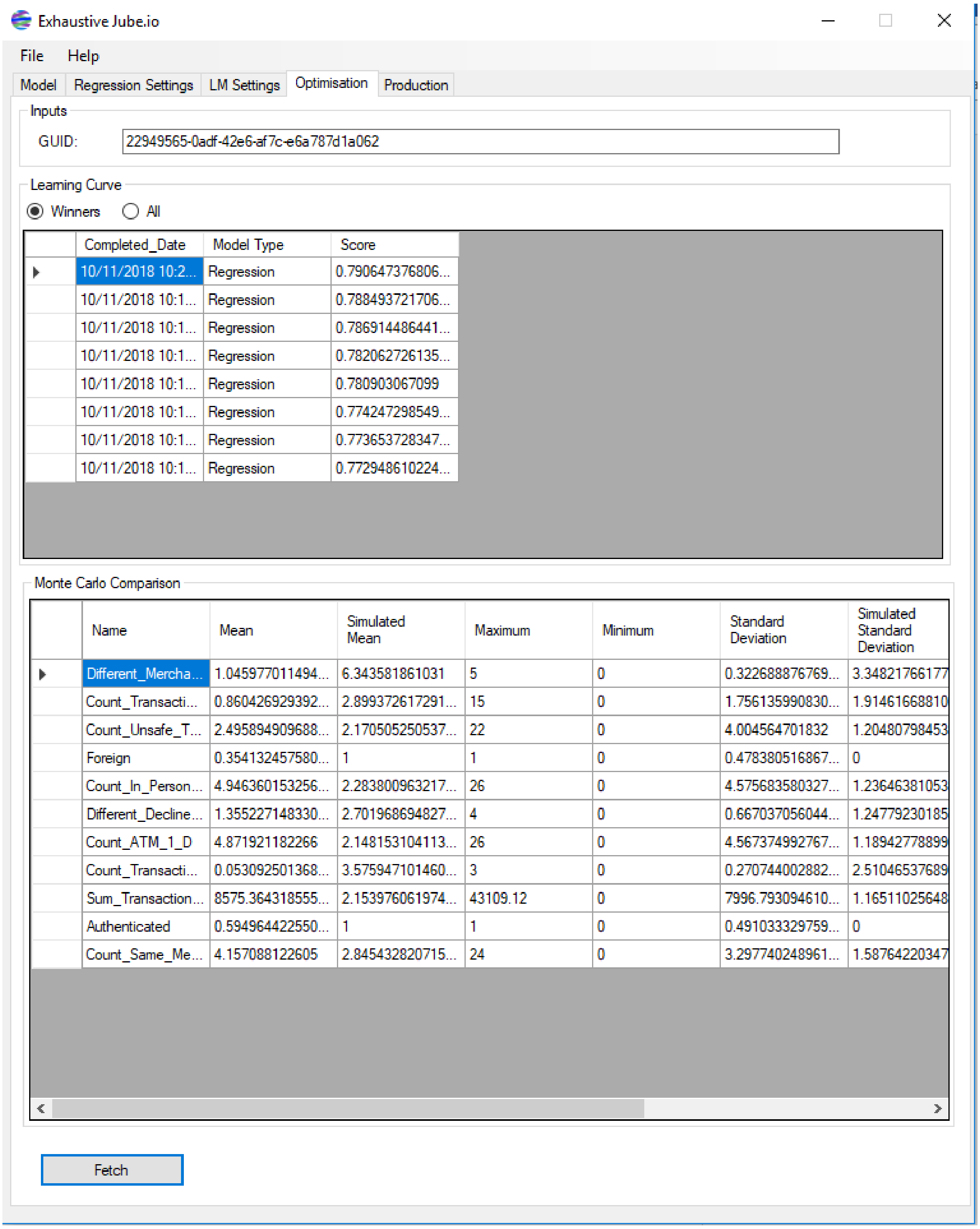


Place the GUID in the GUID textbox:

Navigate to the base of the tab and click the Fetch button, which will now be available:



Upon clicking the fetch button, the model evolution will be returned in the upper grid, with the selected variables being returned in the lower grid:

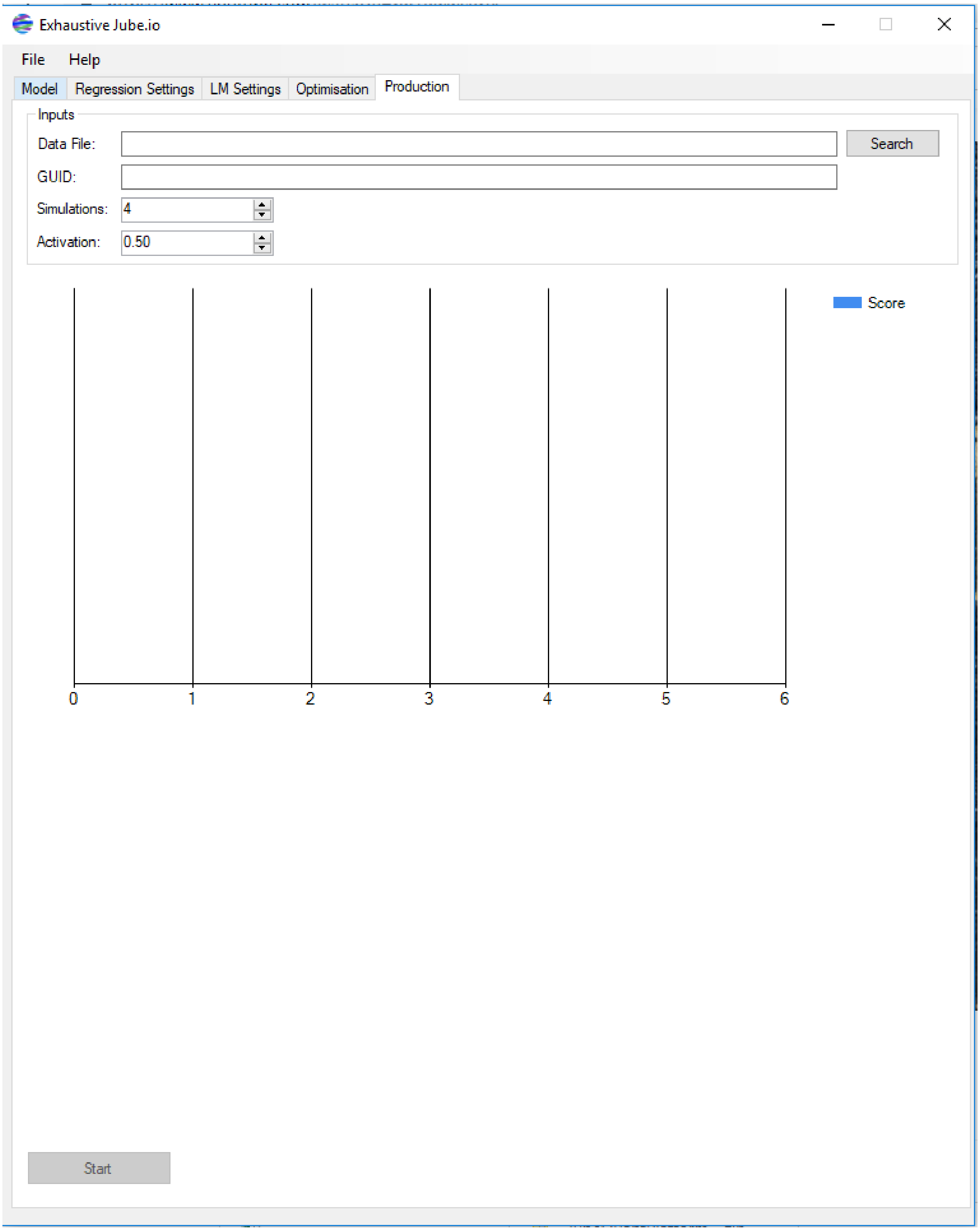


The lower grid, detailing the variable selection, will include statistics and rankings:

- The statistics for each variable calculated before training.
- The statistics derived from Monte Carlo simulation detailing the summary statistics, for each variable, only for the simulations where the score exceeds a given threshold specified in the settings tabs.
- Sensitivity metrics including a ranking and score detailing the most sensitive variable to the least sensitive variable.

The statistics will be produced for the best performing model only. A key requirement is to recall the model against an excel spreadsheet of csv file, so that the model can be used in the day to day operations. Recall can take place by uploading a file, but also via an API (please see Formats document). This example will explore the invocation of the model via file.

To process a file of data through a model, navigate to the Production tab in the Exhaustive Application:

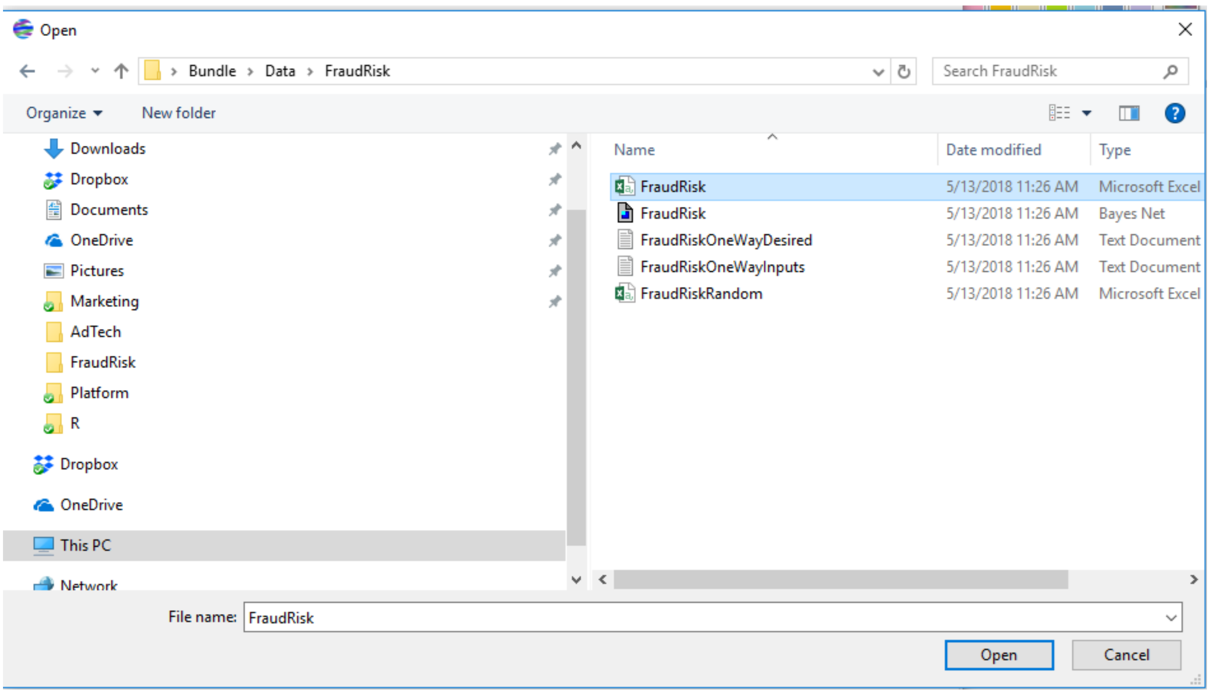


The Production tab takes two parameters. The first parameter is the file that contains records to be processed through the model, being in the same formal as the training dataset albeit without a dependent variable (usually). The second parameter is the GUID of the model to be recalled for each record I the dataset.

Start by clicking the Search button to facilitate the population of the Data File text box with the target file:

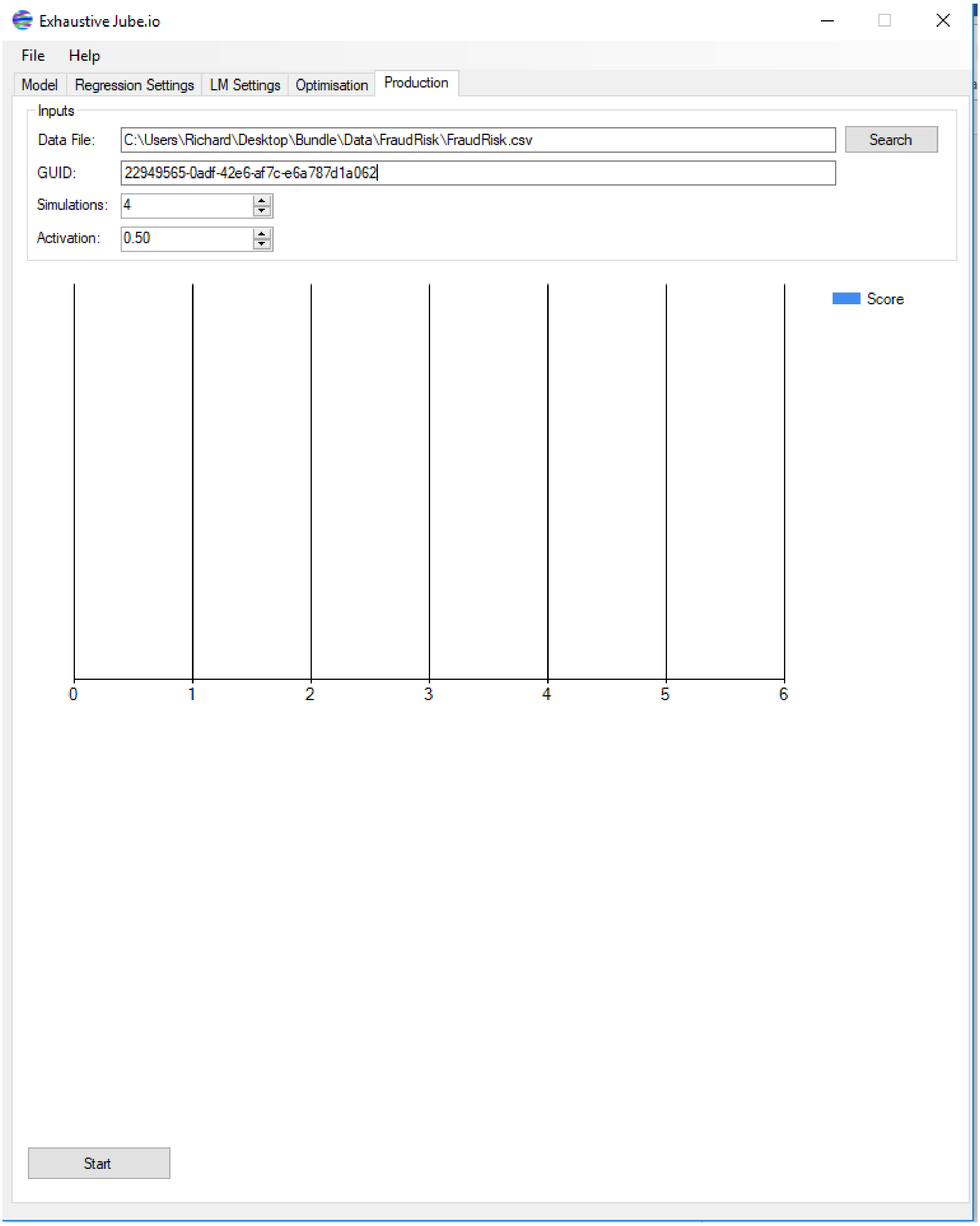
Select the file in the Directory File Explorer Dialog Box, which in this case will be the same file as used for training:

Bundle\Data\FraudRisk\FraudRisk.csv

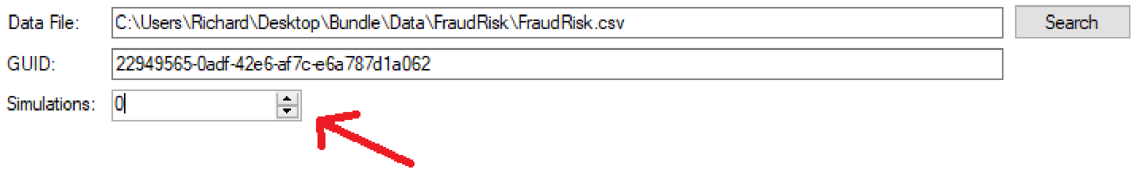


Once the file is selected, pair the GUID by entering it in the GUID textbox as follows:

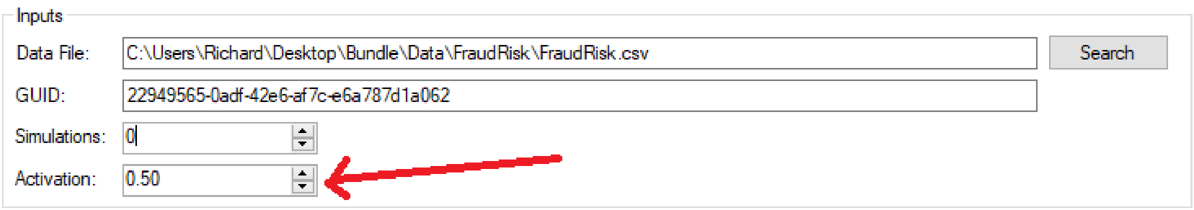
22949565-0adf-42e6-af7c-e6a787d1a062



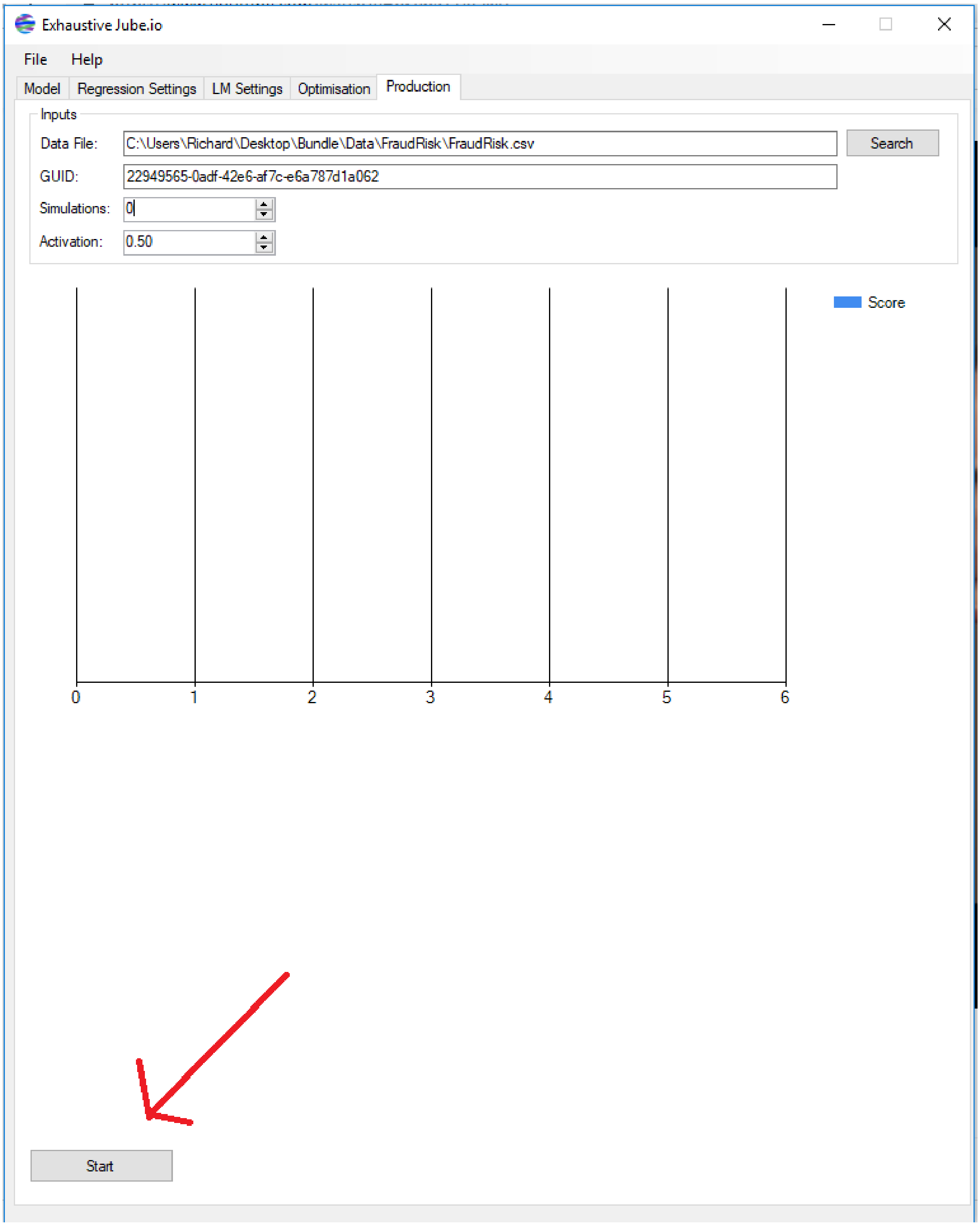
If there are prescriptive variables declared for this model, then it will be fluttered randomly in a triangular distribution as identified from the training dataset during the training process, the Simulations textbox is the number of random simulations to perform. The largest score value will be retained as the optimal and returned to the record as a prescription. In this case, no prescription is required, hence the value is set to zero:



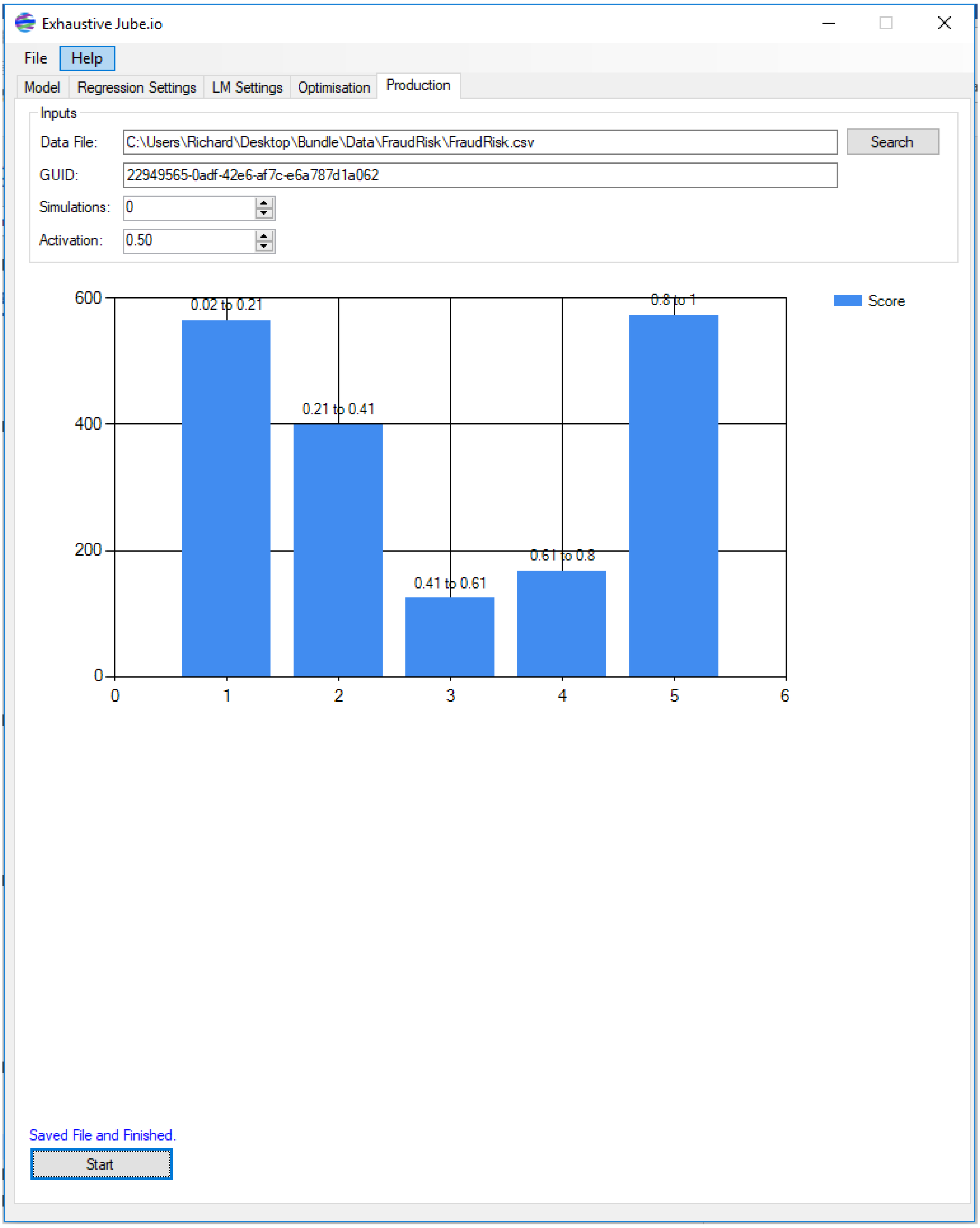
Two columns will be appended to the dataset provided, or a copy of that dataset at least. The first column will be the score returned by the model with the second being a flag which is intended to determine if the record is classified in one direction or another (i.e. 1 or 0). Classification models return as a probability, between 0 and 1, hence values greater than 0.5 would suggest that the record is more likely classified than not:



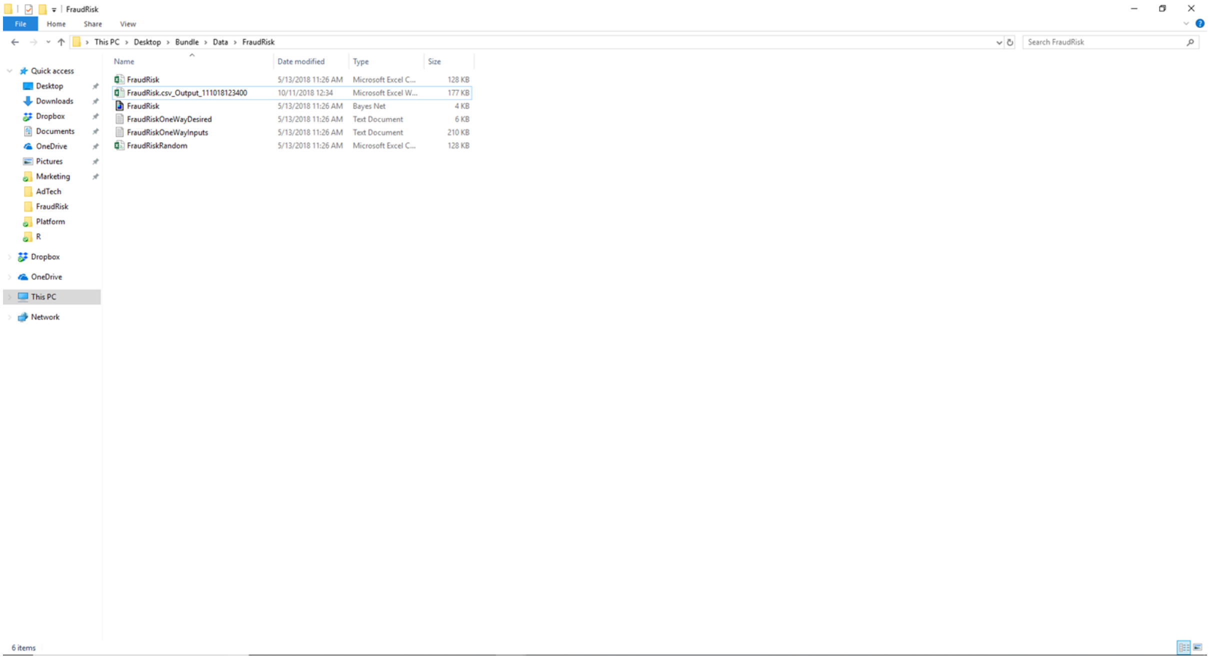
Upon selecting the values for model recall, click the Start button at the base of the Production tab:

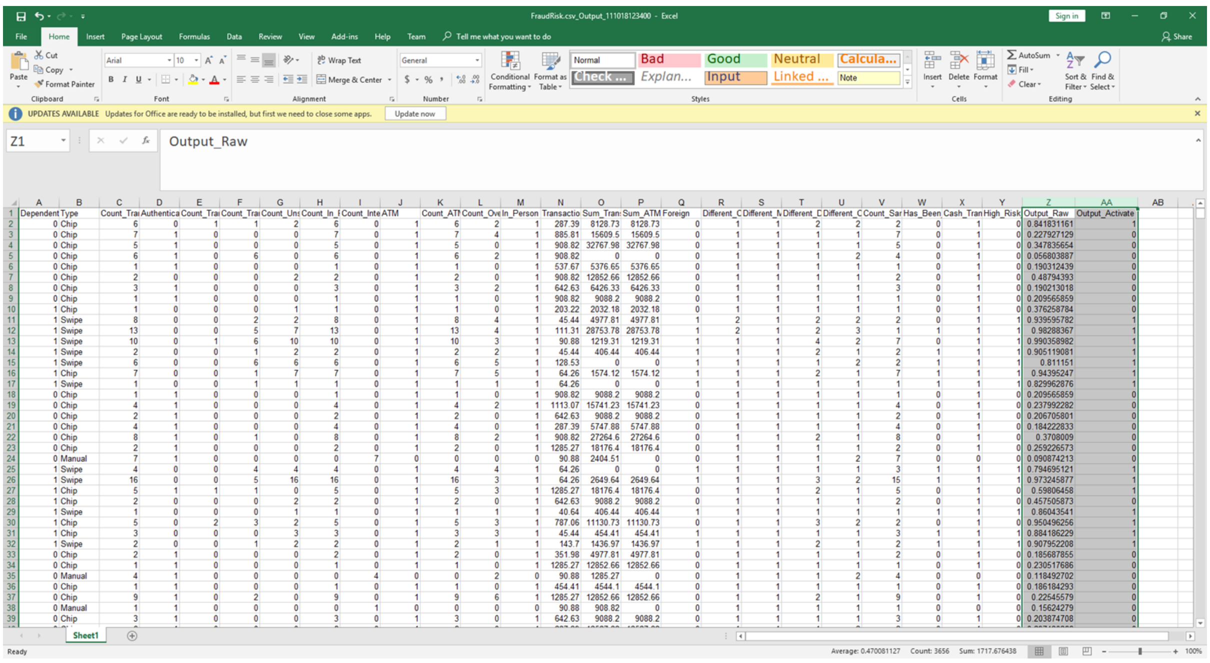


Upon clicking the start button the file will be loaded with each record being processed through the model and returning a score. The status of processing will be written out to a status bar during processing. Upon completion of processing, a histogram of the scores achieved will be created:



A file will be created in the same directory as the original dataset, copied and appended with the score and an activation flag:





In the event that a prescription variable has been specified, this value will be updated for each record.

Module 15: Deep Learning with H20

H20 is an external server-based software application that presents a variety of machine learning algorithms. The machine learning algorithms available do not materially differ from those already presented and freely available in R. H20 provides several useful features for deep learning though:

- Compression of data while processing, in a hex format. This makes the memory requirements less burdensome, keeping in mind that R keeps data frames in memory otherwise and;
- Compatibility with a GPU to facilitate Deep Neural Networks and Deep Learning.

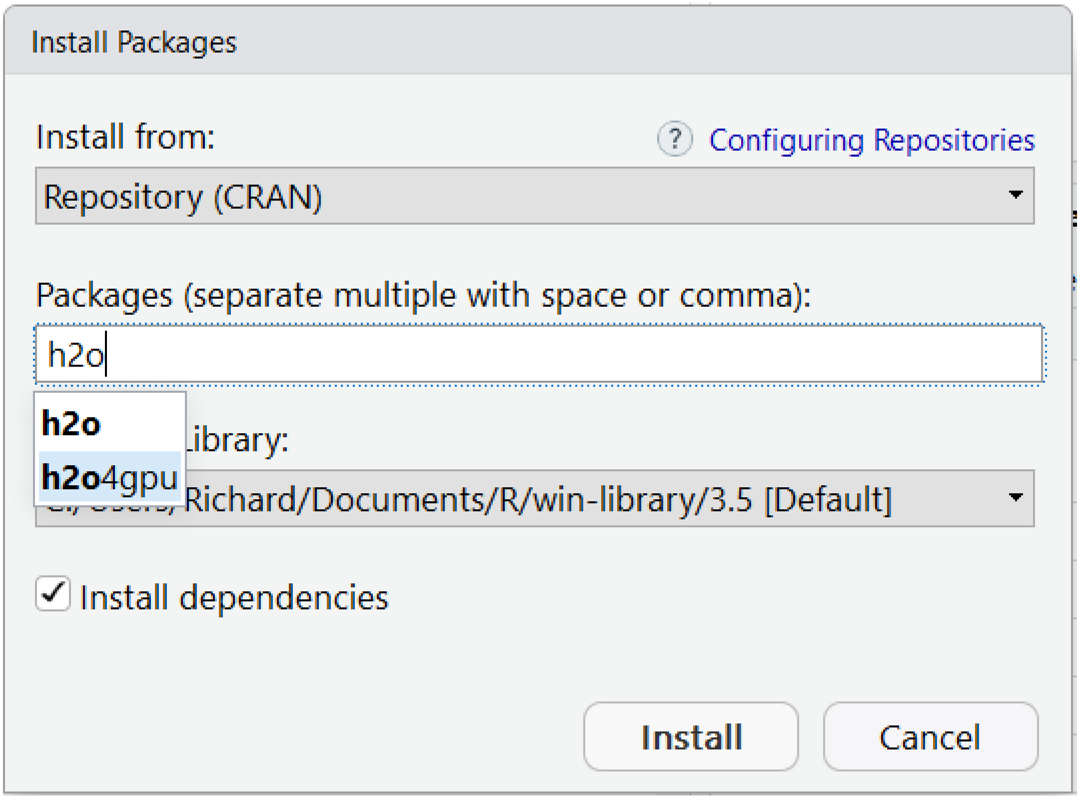
H20 has exceptionally well coupled with R, and while processing will take place in the H20 server, it is as though it is taking place within RStudio. H20 exists in the mix of tools predominately because of its ability to use a GPU for deep learning as well as providing granular control over cross validation and activation functions.

H20 is fundamentally an API led platform and R is just one tool that can make use of the tool via API. H20 has its own tool, called Flow, that can invoke these API's and make for a self-service user interface, setting a low bar to create models. These procedures will use Flow to create a familiarity with H20, before seeking to replicate the processes with R commands, which is how most users will tend to interact with H20.

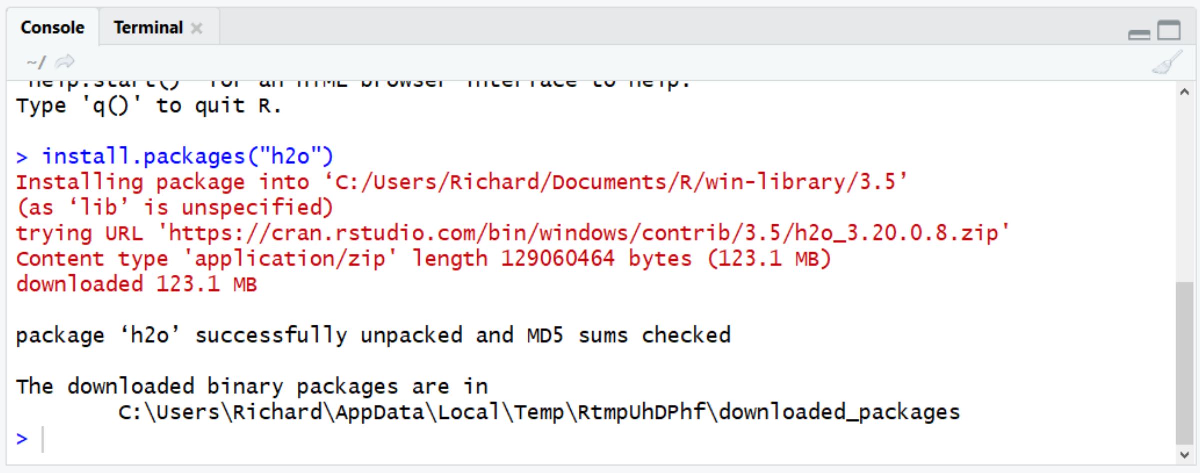
Procedure 1: Install H2O package, instantiate and browse to the Flow User Interface

Even though H2O is server software and runs externally to R, it can be installed and initialised from with R. Installing the entire H2O server is no more complex than installing any other R package.

To install H20, use RStudio and begin by installing the H20 package:

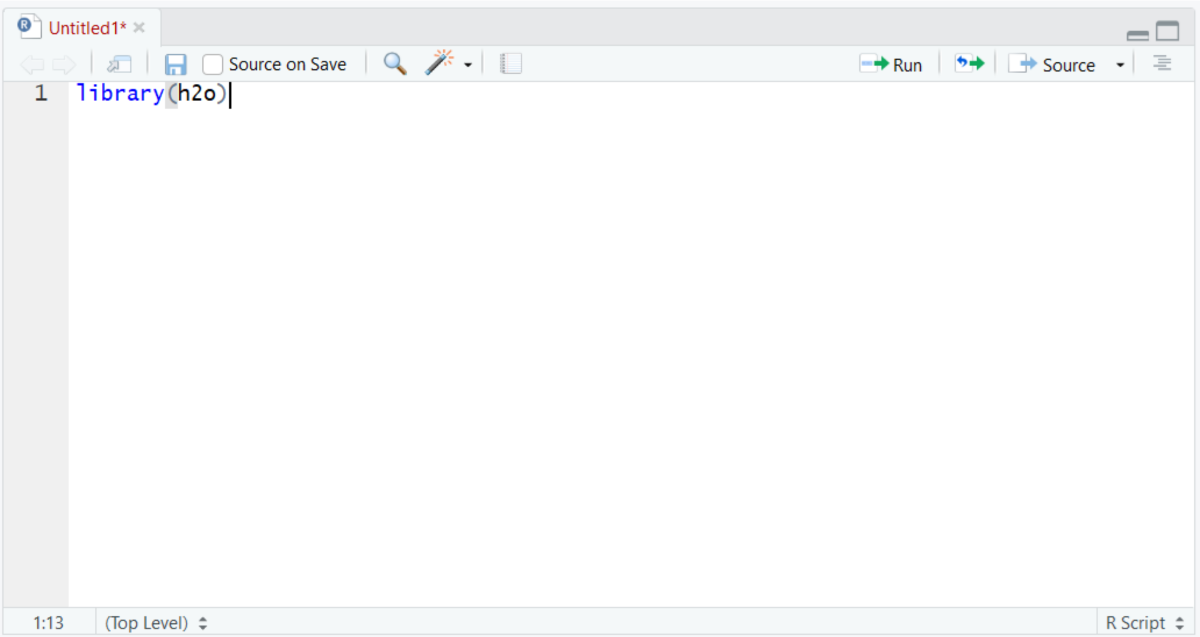


Wait for the installation to complete, although this will take a little bit longer than most packages as it is big:

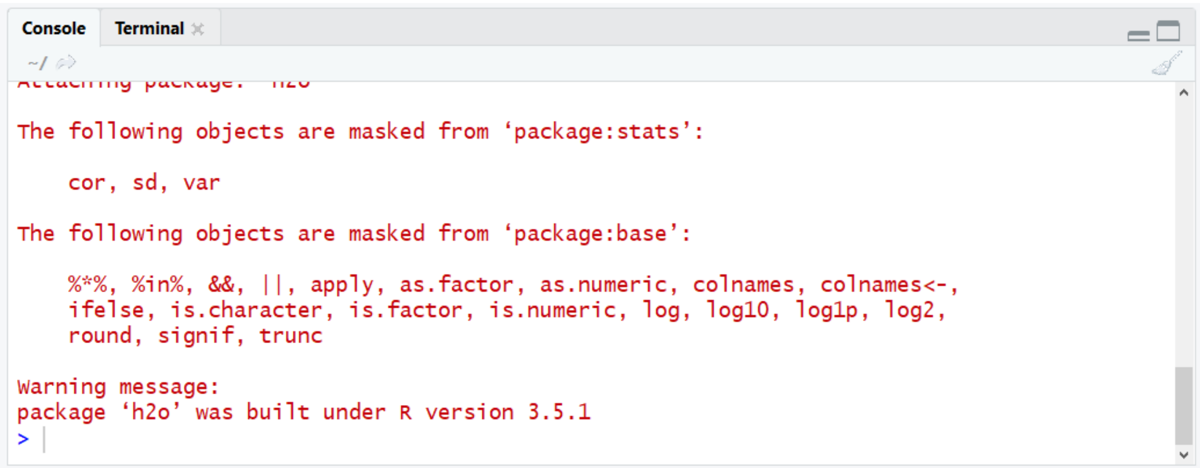


Load the H2O package by typing:

library(h2o)

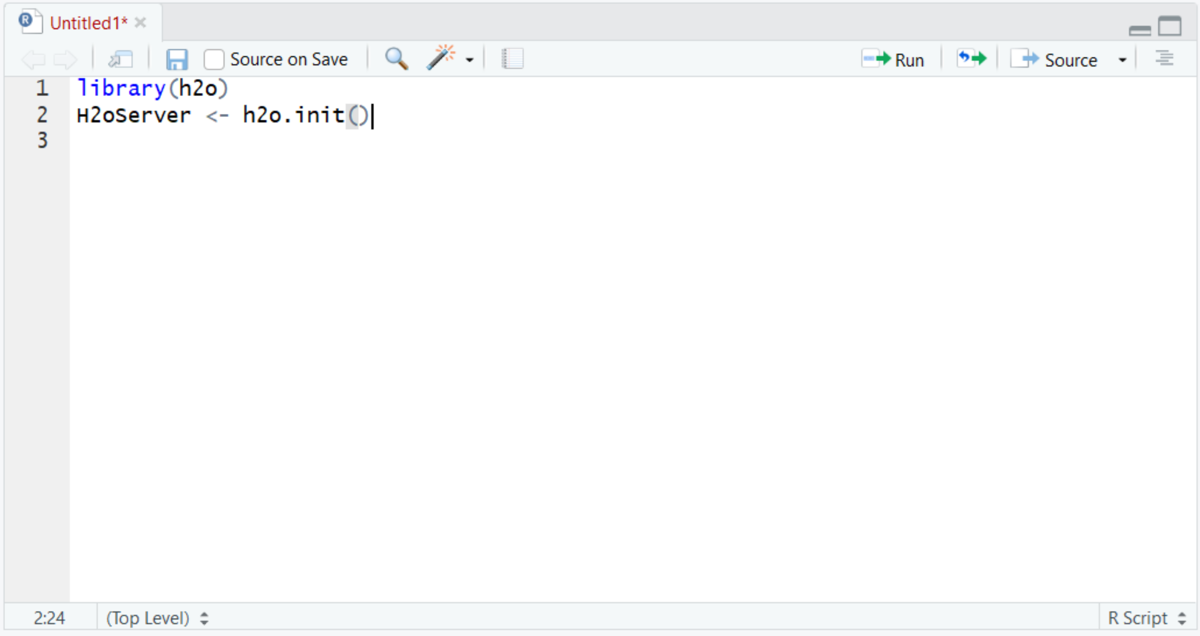


Run the line of script to console:

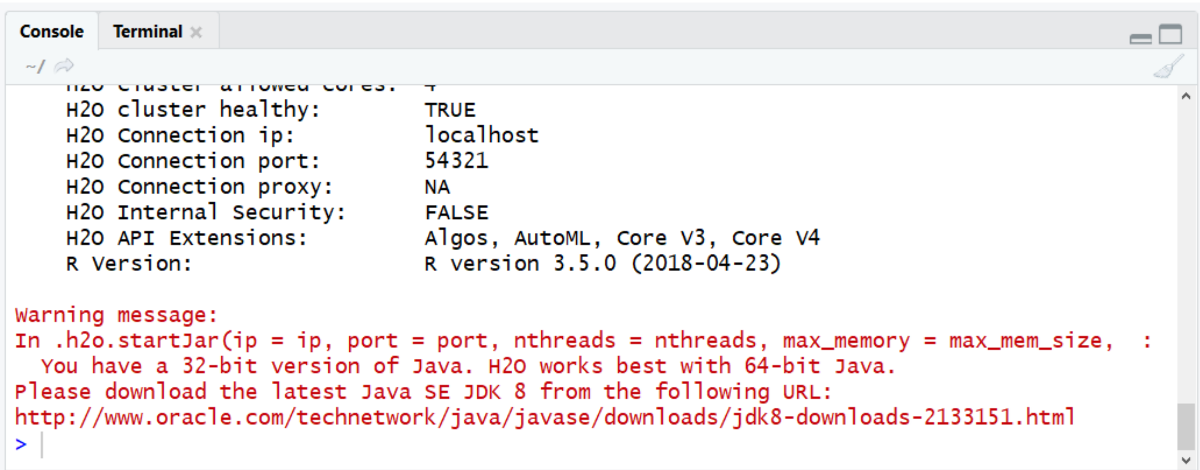


The H2O server needs to be started externally, but this can be achieved through a helper function available to the H2O library. To start the H2O server, use the h2o.init function with the default parameters (i.e. no parameters):

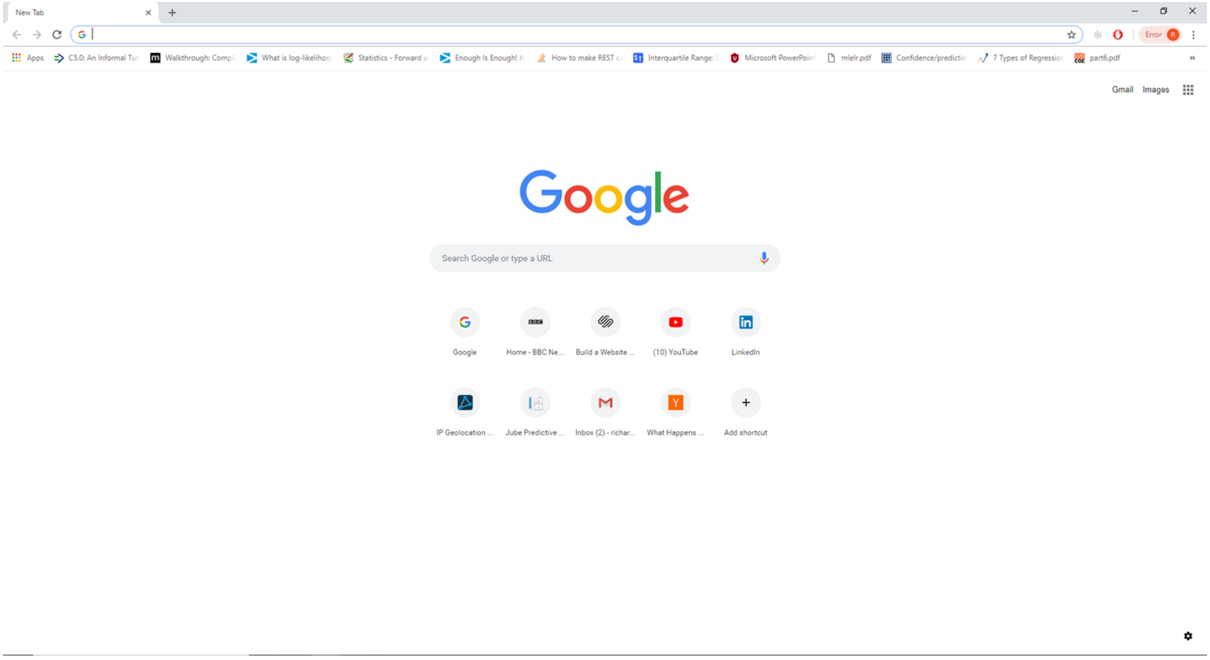
H2oServer <- h2o.init()



Run the line of script to console and wait for confirmation to be provided that the h2o server has been started externally to R:

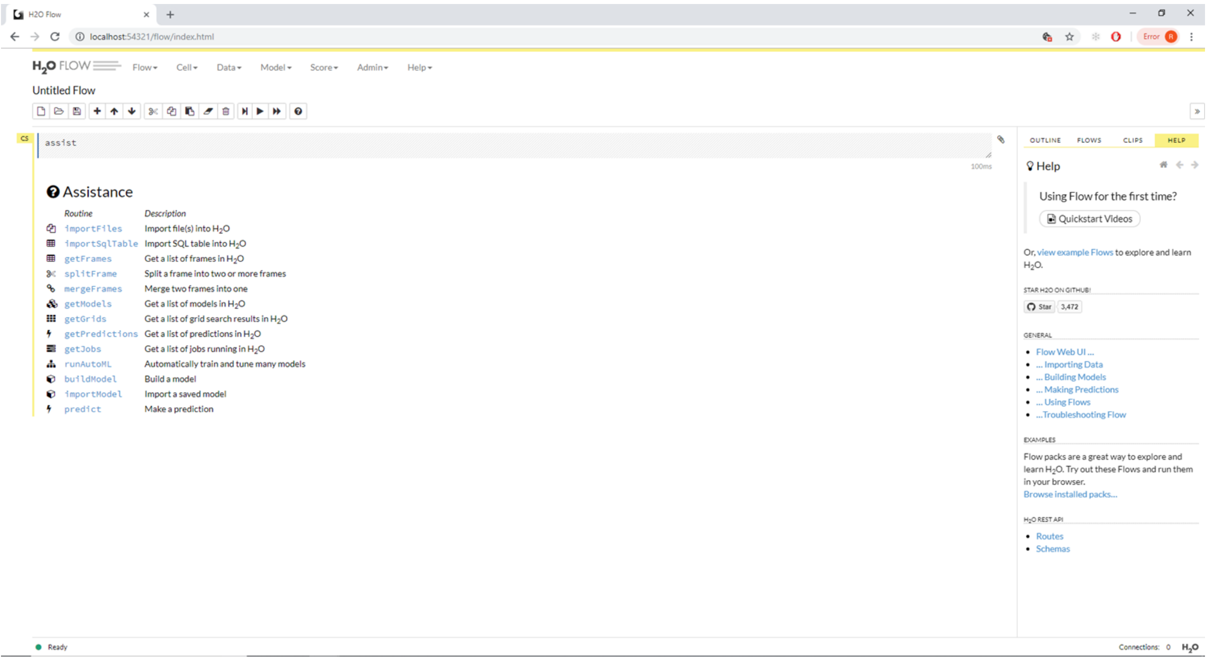


The h2o server acts as a web server which serves up the Flow application. To navigate to the Flow application, open a browser such as Chrome:



Navigate to the URL:

http://localhost:54321



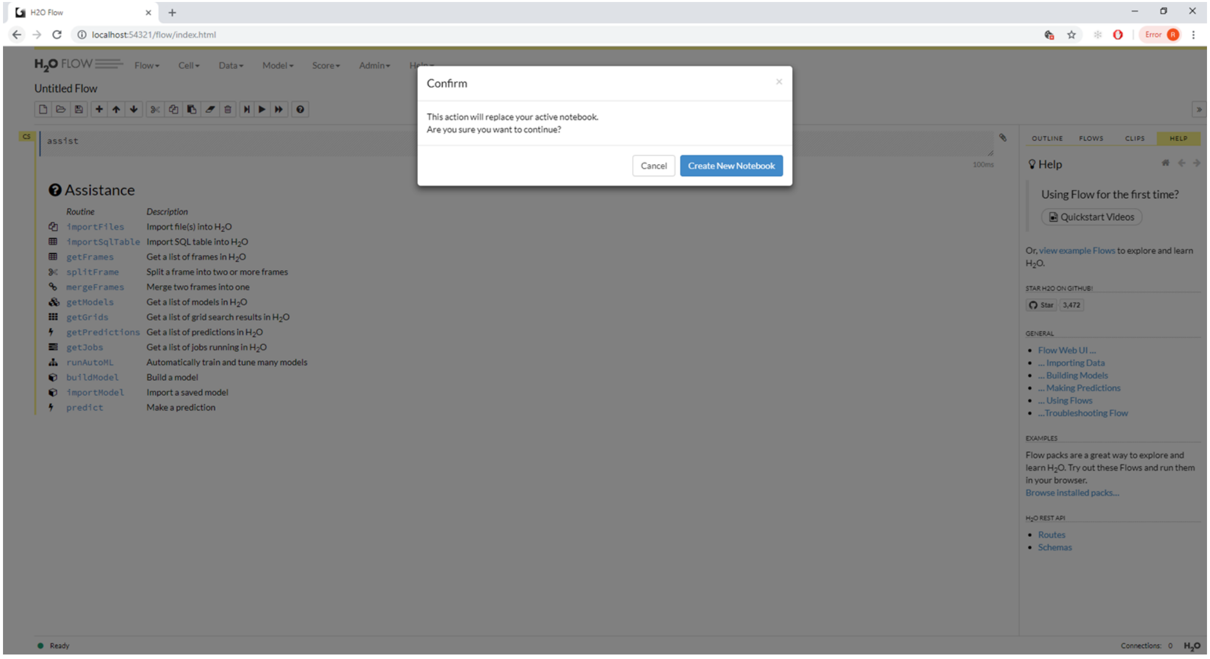
The H2o server is now installed and available for use via the Flow user interface, API or R commands.

Procedure 2: Loading Data into H2O with Flow

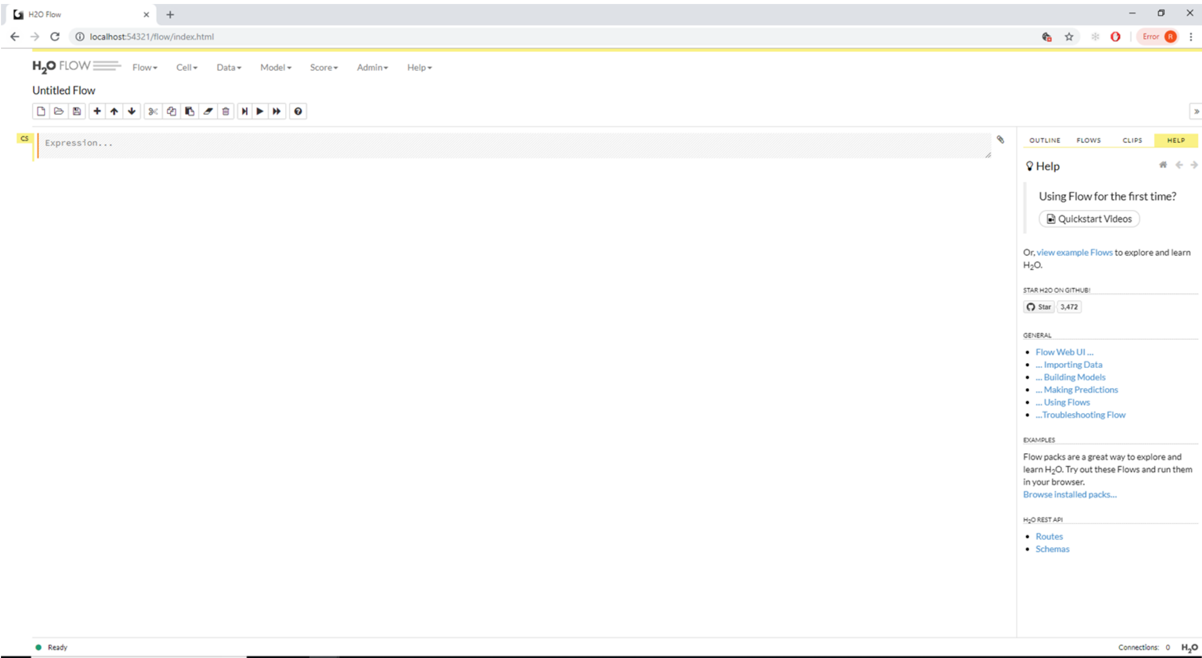
In this example a logistic regression model will be created, using Flow, achieving the same results as achieved in the GLM functions of R and Exhaustive.

In the Flow user interface, start by navigating:

Flow >>> New Flow



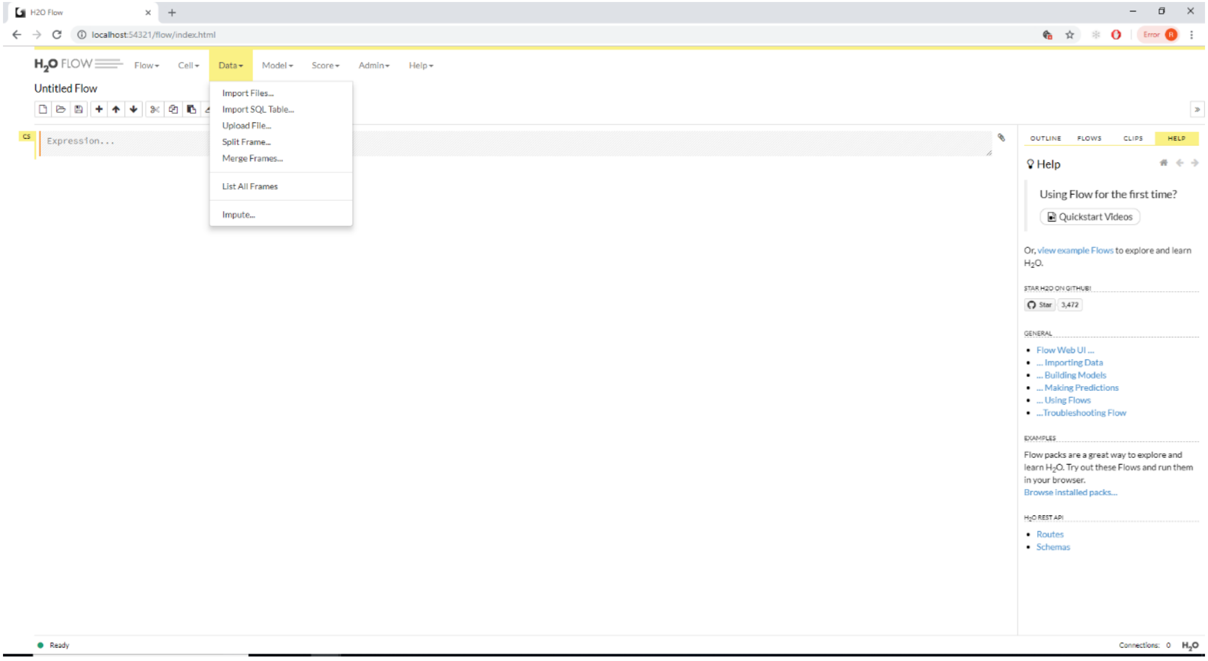
If prompted to create a new workbook, affirm this:



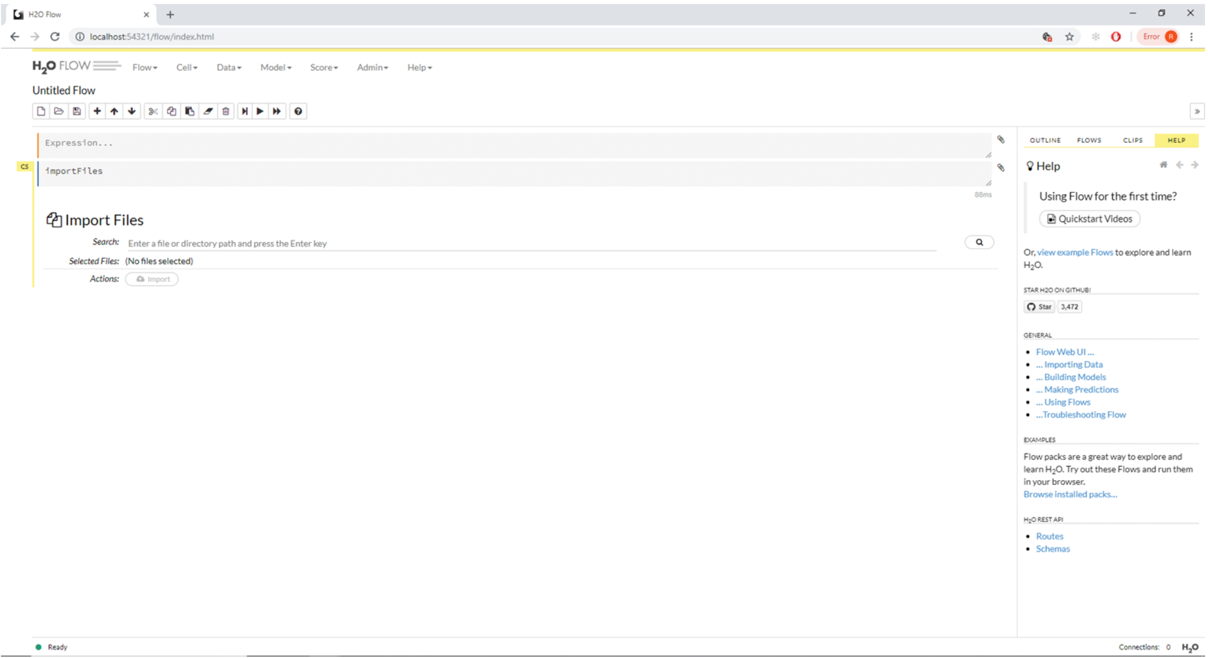
To add a cell for the importing of data, navigate to:

Data >>> Import Files

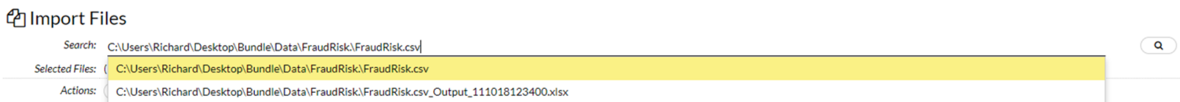
JUBE



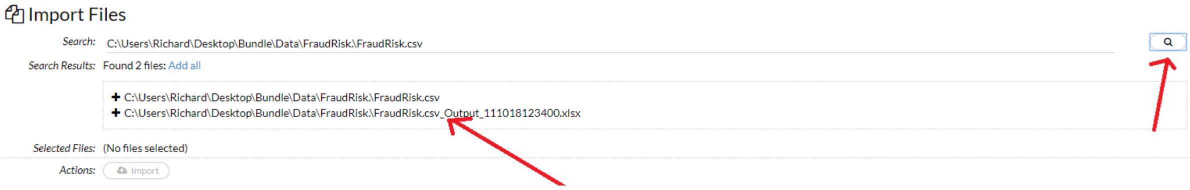
It can be seen that Import Files Cell has been added to the Flow:



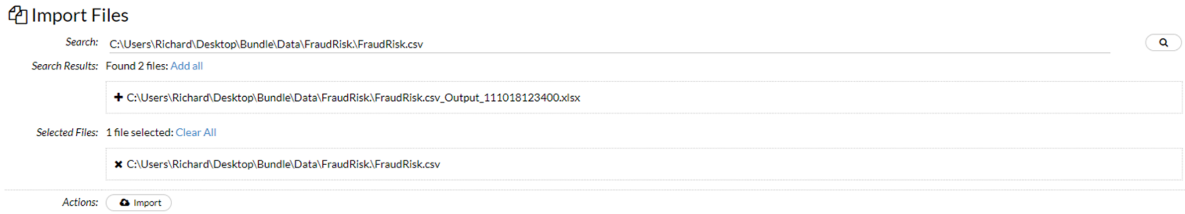
In the Search dialog box, enter the location of the FraudRisk.csv file until a drop down is populated, for example:



Click on the Search Icon to bring back the contents of this directory:



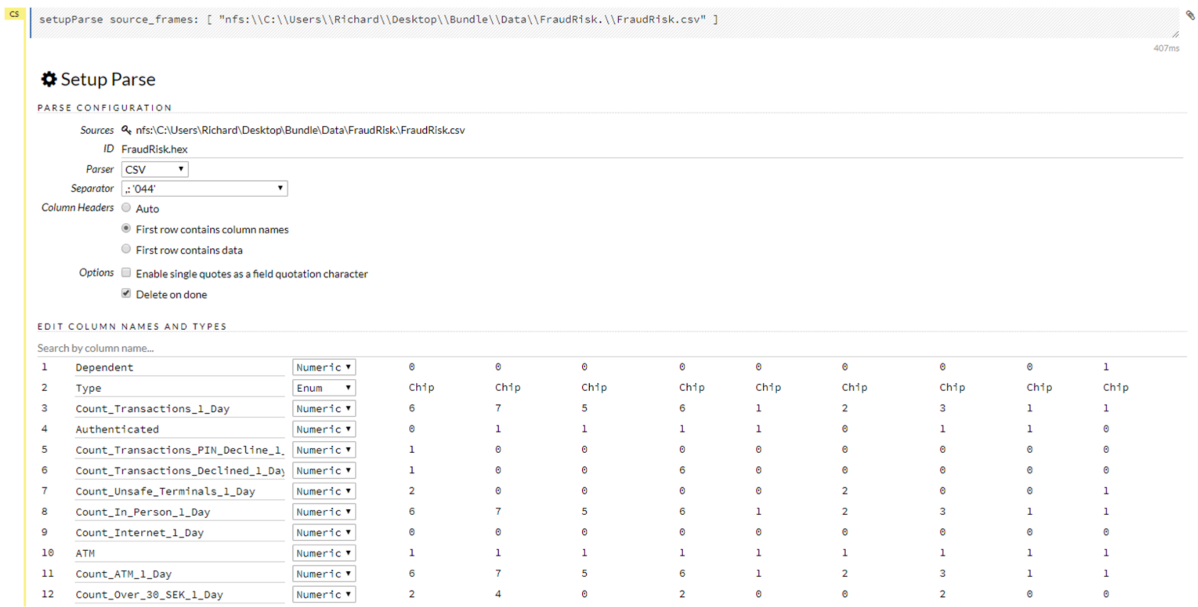
Click on the file or plus sign to add the file to the cell:



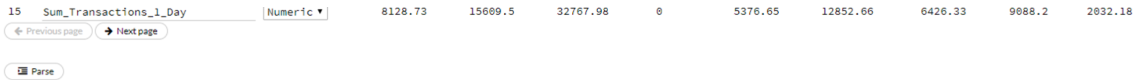
Click the Import Button to import the file to H2O:

Note that the file is not parsed to the H2O column compressed format, known as Hex. To achieve parsing, simply click the button titled 'Parse These Files':

The next screen allows for the specification and data types to be more robustly configured. In this example, a cursory check to ensure that the data types are correct is sufficient:



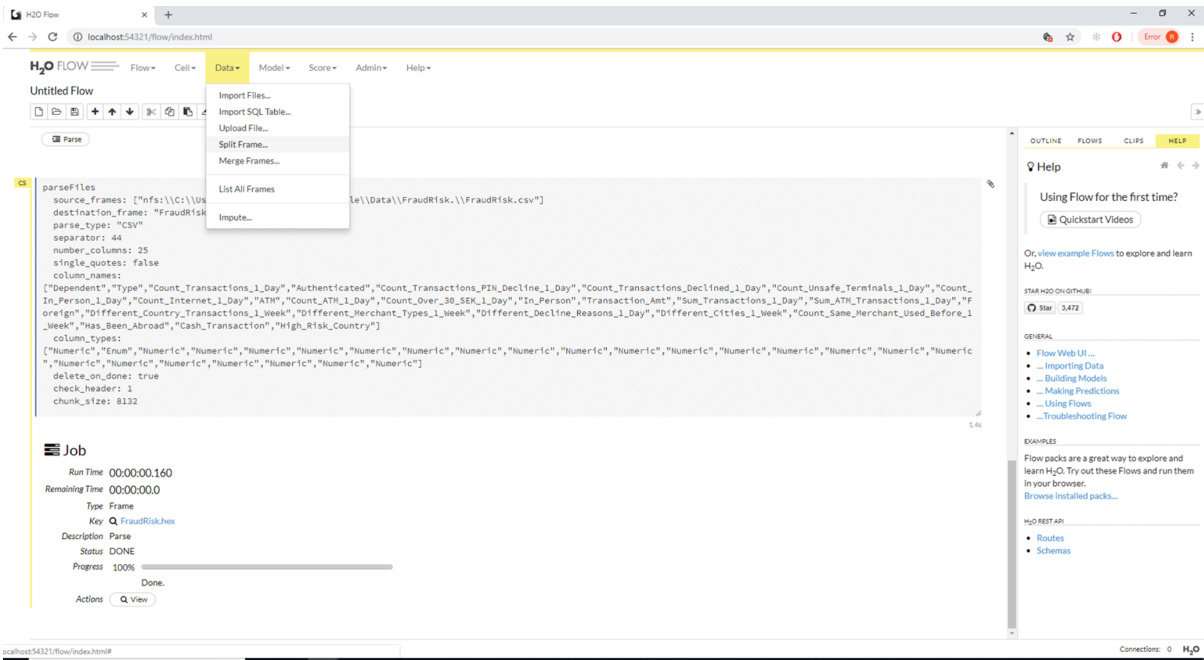
Upon satisfaction, click parse to mount the dataset in H20 as Hex:



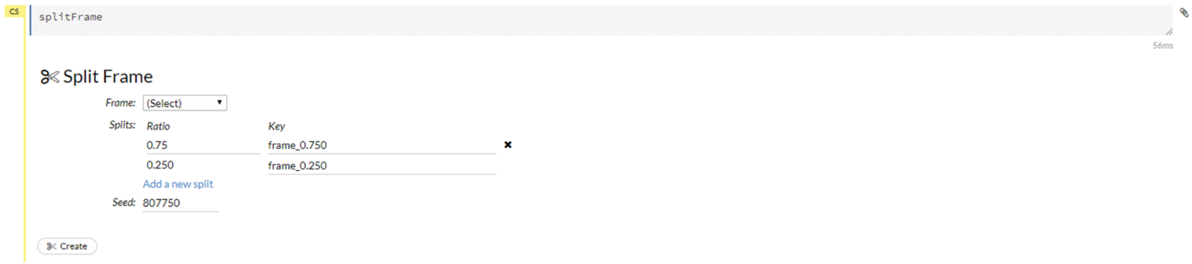
A background job will start the process of transforming the data from FraudRisk.csv to the H2O hex format:

H2O supports the concept of training and validation datasets robustly, henceforth the hex file needs to be split into training and validation. To split a Hex frame, navigate to:

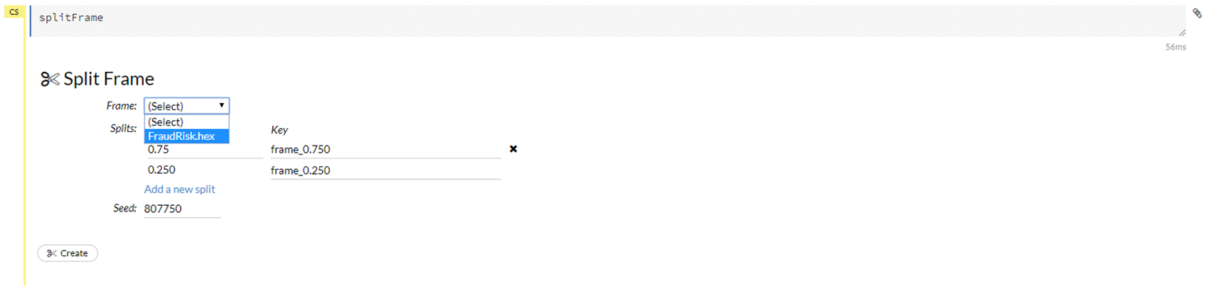
Data >>> Split Frame



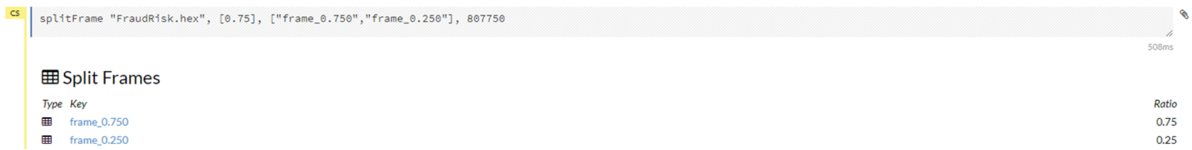
Click on the menu item to create the split data frame cell:



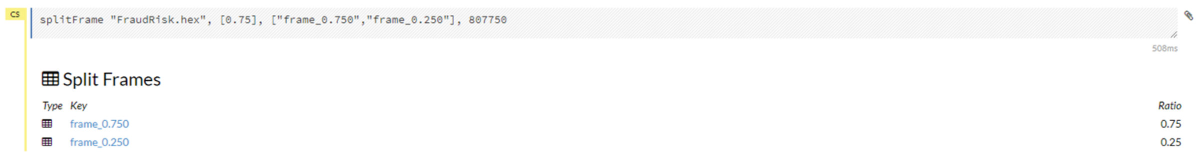
Select the frame to be split, in this case FraudRisk.hex:



The default frame split is 75% by 25%, confirm this by clicking the Create button:



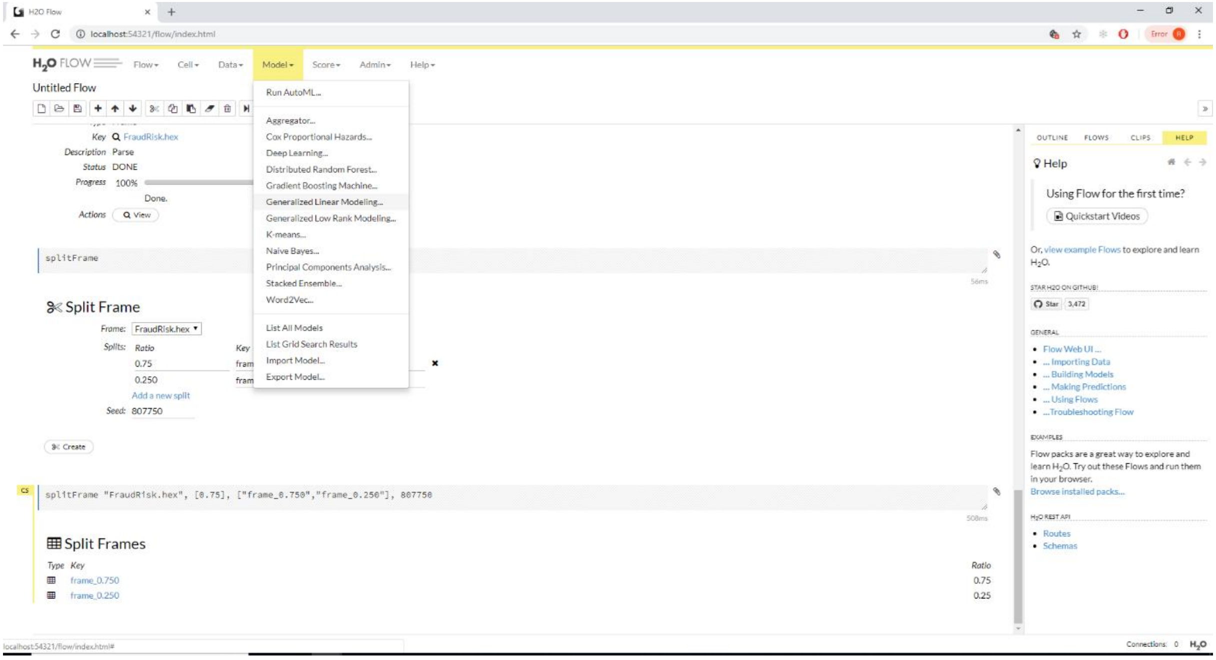
There now exists two frames in the flow, the smaller of which will be used for validation:



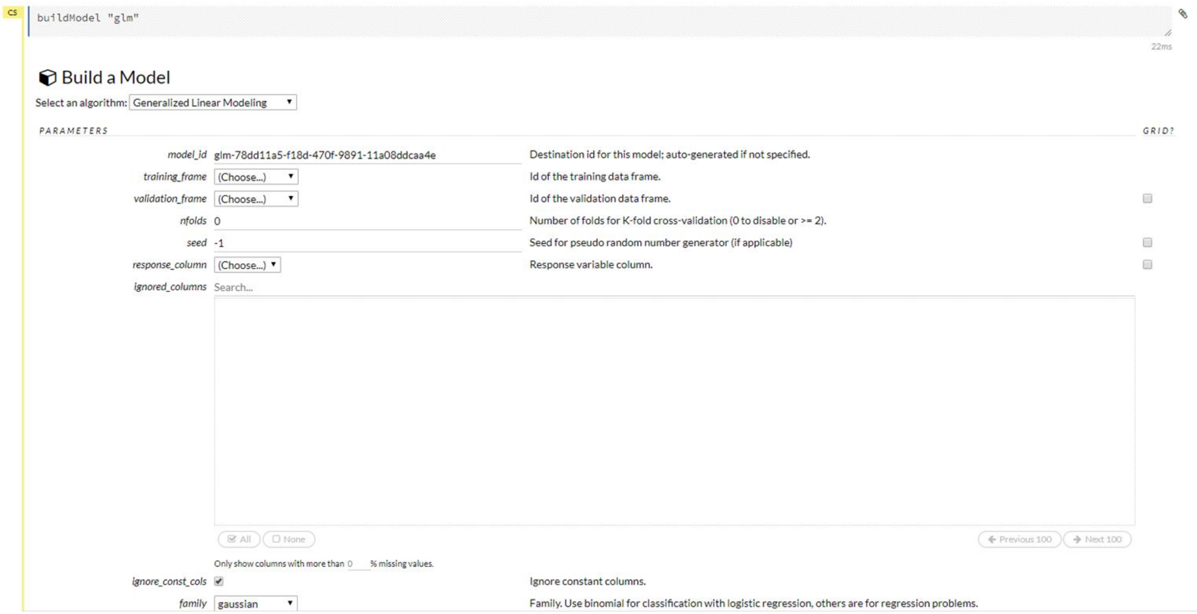
Procedure 3: Creating a Logistic Regression model in H2O (GLM)

With the data loaded, a model now needs to be trained. Navigate to Models to see the available algorithms:

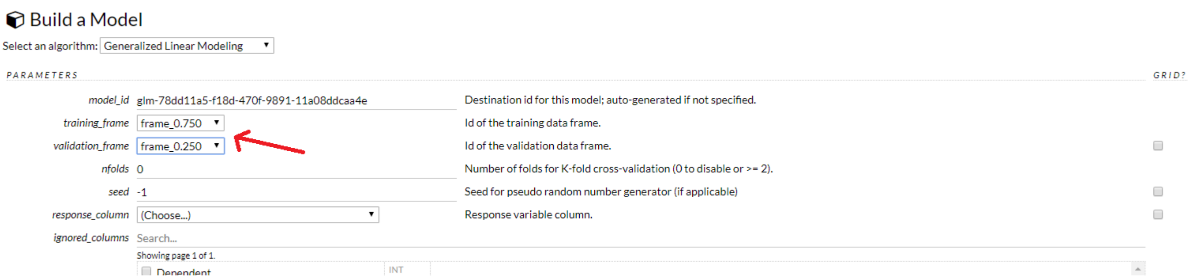
Models



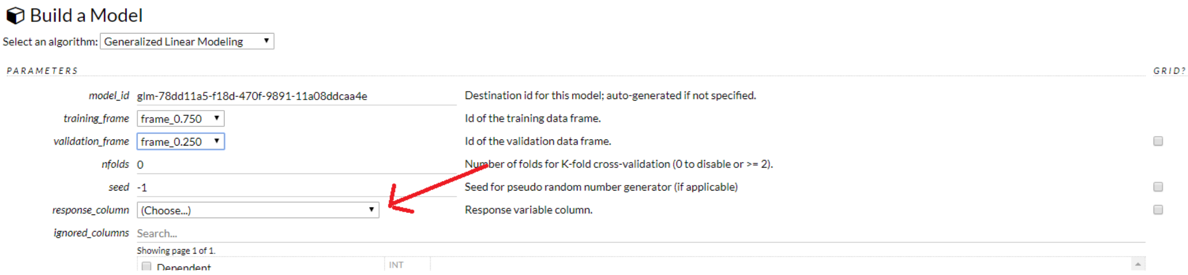
In this case, the algorithm is Generalised Linear Modelling (this is Logistic Regression). Click this model to create the cell in flow:



There are a multitude of parameters that are quite outside the scope of this document, for the purposes of this document, simply specify the Training and Validation Hex sets:

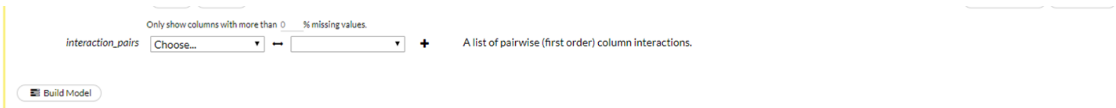


Thereafter, specify the dependent variable, known as the Response Column in H2O:



In this case the Dependent Variable is titled as the same:

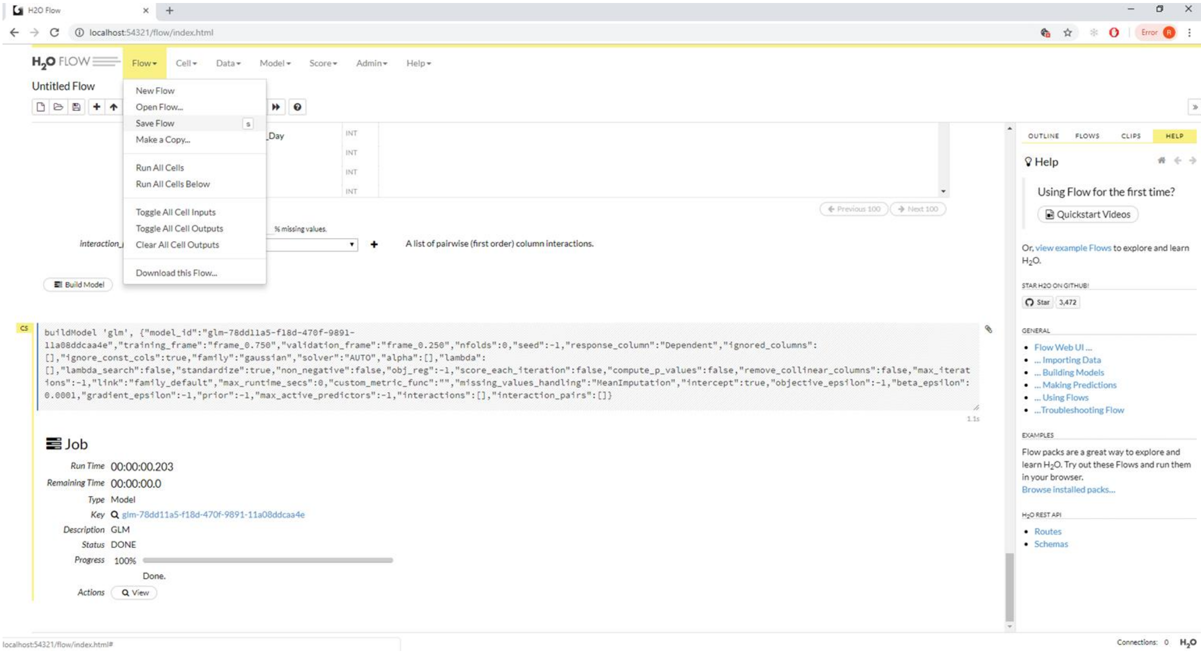
Scroll to the base of the cell and click Build Model to initiate the training process:



The training process will begin with progress being written out to a newly created job cell:

At this stage a Logistic Regression model has been created. It is a good idea to save the flow by navigating:

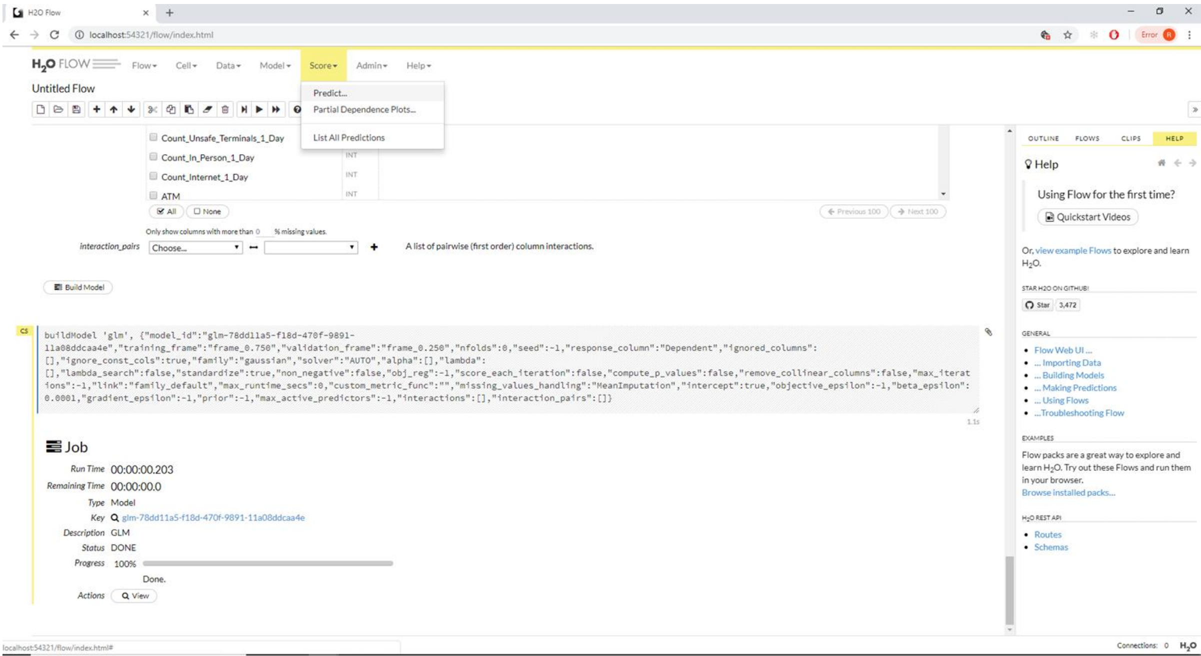
Flow >>> Save Flow



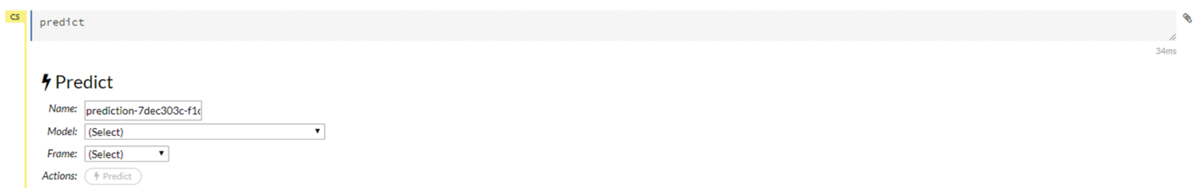
Procedure 4: Recalling a Logistic Regression model with Flow

To recall this logistic regression model from flow, navigate to:

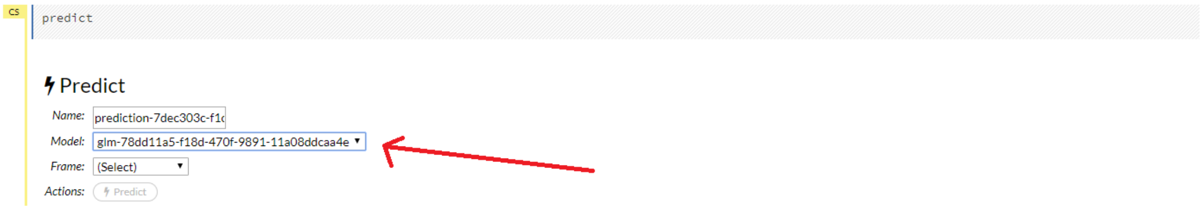
Scores >>> Predict



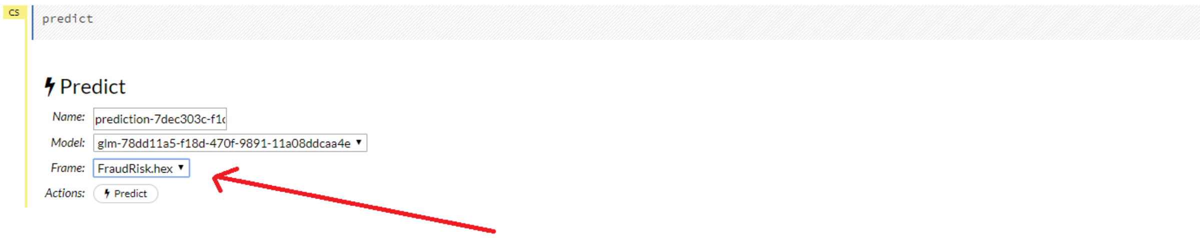
The predict cell will be added to the Flow:



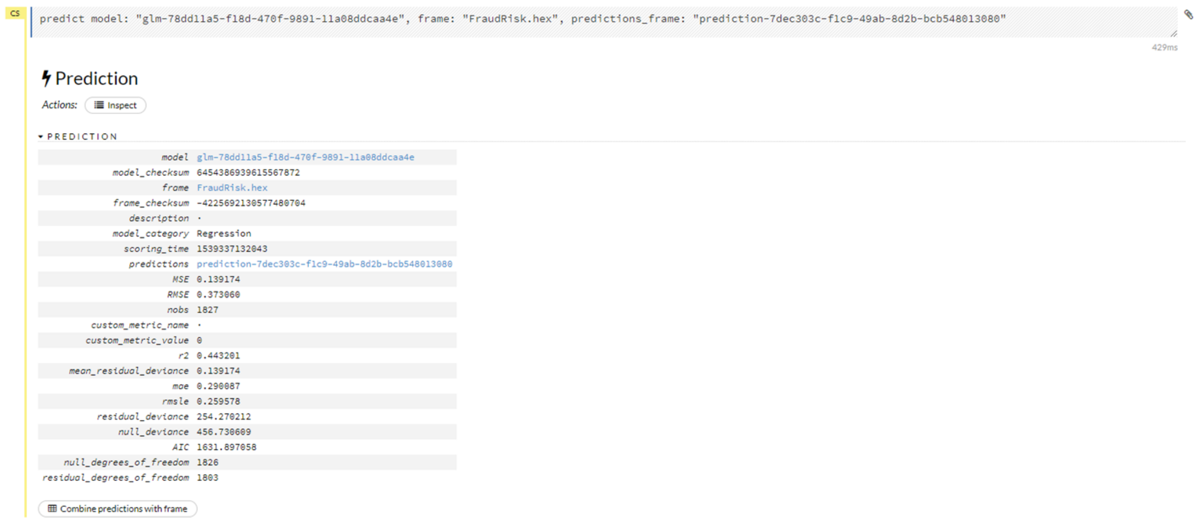
The recall of the model may assume that a new frame has been created in flow, but for this example, the validation frame will be recalled via the logistic regression, trained, model. Firstly, set the model to recall:



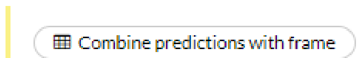
Thereafter, select the data frame to process through the model:



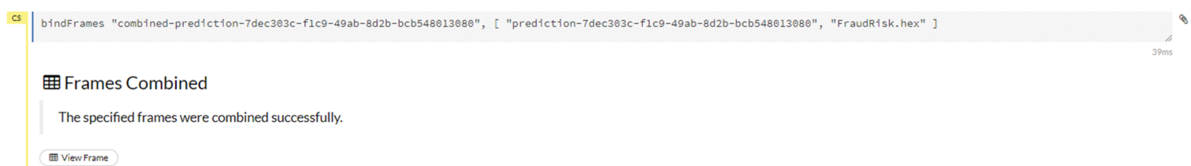
Upon selecting the input parameters, click the predict button to complete the prediction. A cell detailing the output will be created:



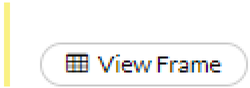
It is sensible at this stage to combine the predictions with the original dataset. To combine the predictions with the original dataset, simply click the Combine Predictions with Frame button:



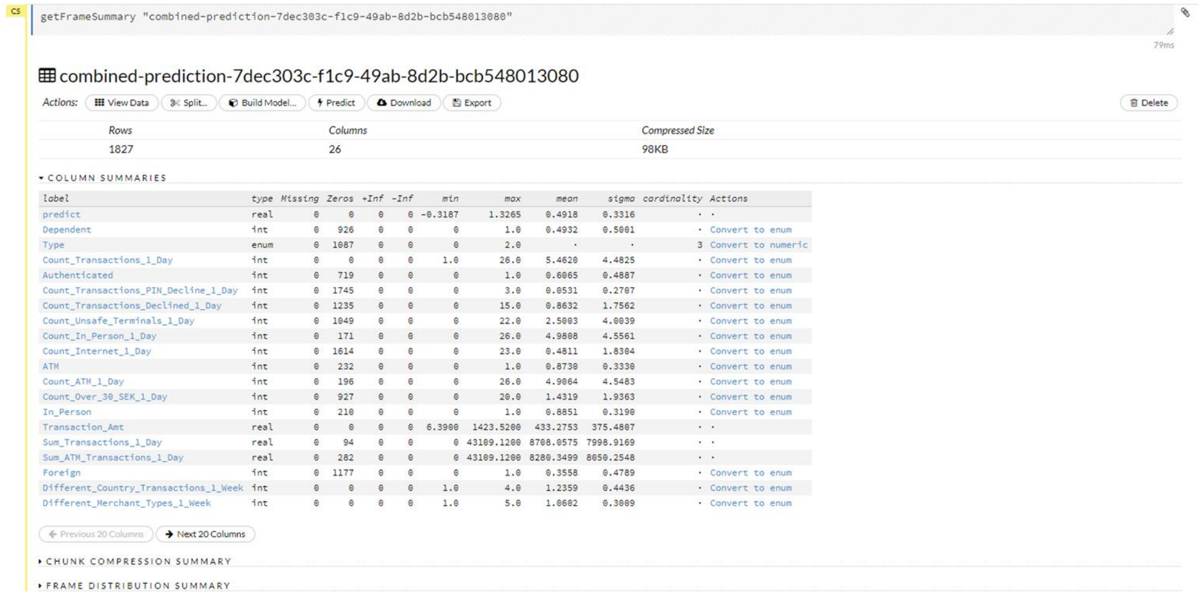
Upon combining the predictions with the original dataset, the dataset will be available for download:



To interact with the newly created data frame click on the View Frame button:



The View Frame functionality provides for the downloading and further manipulation of the data frame:



getFrameSummary "combined-prediction-7dec303c-f1c9-49ab-8d2b-bcb548013080"

79ms

combined-prediction-7dec303c-f1c9-49ab-8d2b-bcb548013080

Actions: View Data | Split... | Build Model... | Predict | Download | Export | Delete

Rows	Columns	Compressed Size
1827	26	98KB

COLUMN SUMMARIES

label	type	Missing	Zeros	+Inf	-Inf	min	max	mean	sigma	cardinality	Actions
predict	real	0	0	0	0	-0.3187	1.3265	0.4918	0.3316	·	·
Dependent	int	0	926	0	0	0	1.0	0.4932	0.5001	·	Convert to enum
Type	enum	0	1087	0	0	0	2.0	·	·	3	Convert to numeric
Count_Transactions_1_Day	int	0	0	0	0	1.0	26.0	5.4620	4.4825	·	Convert to enum
Authenticated	int	0	719	0	0	0	1.0	0.6065	0.4887	·	Convert to enum
Count_Transactions_PIN_Decline_1_Day	int	0	1745	0	0	0	3.0	0.0531	0.2707	·	Convert to enum
Count_Transactions_Declined_1_Day	int	0	1235	0	0	0	15.0	0.8632	1.7562	·	Convert to enum
Count_Unsafe_Terminals_1_Day	int	0	1049	0	0	0	22.0	2.5003	4.0039	·	Convert to enum
Count_In_Person_1_Day	int	0	171	0	0	0	26.0	4.9808	4.5561	·	Convert to enum
Count_Internet_1_Day	int	0	1614	0	0	0	23.0	0.4811	1.8304	·	Convert to enum
ATM	int	0	232	0	0	0	1.0	0.8730	0.3330	·	Convert to enum
Count_ATM_1_Day	int	0	196	0	0	0	26.0	4.9064	4.5483	·	Convert to enum
Count_Over_30_SEK_1_Day	int	0	927	0	0	0	20.0	1.4319	1.9363	·	Convert to enum
In_Person	int	0	210	0	0	0	1.0	0.8851	0.3190	·	Convert to enum
Transaction_Amt	real	0	0	0	0	6.3900	1423.5200	433.2753	375.4807	·	·
Sum_Transactions_1_Day	real	0	94	0	0	0	43109.1200	8708.0575	7998.9169	·	·
Sum_ATM_Transactions_1_Day	real	0	282	0	0	0	43109.1200	8280.3499	8050.2548	·	·
Foreign	int	0	1177	0	0	0	1.0	0.3558	0.4789	·	Convert to enum
Different_Country_Transactions_1_Week	int	0	0	0	0	1.0	4.0	1.2359	0.4436	·	Convert to enum
Different_Merchant_Types_1_Week	int	0	0	0	0	1.0	5.0	1.0602	0.3009	·	Convert to enum

Previous 20 Columns | Next 20 Columns

CHUNK COMPRESSION SUMMARY

FRAME DISTRIBUTION SUMMARY

The process thus far uses the Flow user interface to create something akin to a script, where it is the flow tool that is sending instructions to the H2O API. It would be far less cumbersome to use R scripting to achieve such flows.

Procedure 5: Loading Data into h2O with R

Start by loading the FraudRisk.csv file into R using readr:

library(readr)

FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
```

Run the block of script to console:

```
Please download the latest Java SE JDK 8 from the following URL:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
> library(readr)
> FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
>
```

The training process will make use of a test dataset and a sample dataset. The preferred method to randomly split a dataframe is to create a vector which comprises random values, then append this vector to the dataframe. Using Vector sub setting, data frames will be split based on a random value.

Start by observing the length of the dataframe by typing (on any dataframe variable):

length(FraudRisk$Dependent)

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
```

Run the line of script to console:

```
> library(readr)
> FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> length(FraudRisk$Dependent)
[1] 1827
>
```

Having established that the dataframe has 1827 records, use this value to create a vector of the same size containing random values between 0 and 1. The RunIf function is used to create vectors or a prescribed length with random values between a certain range:

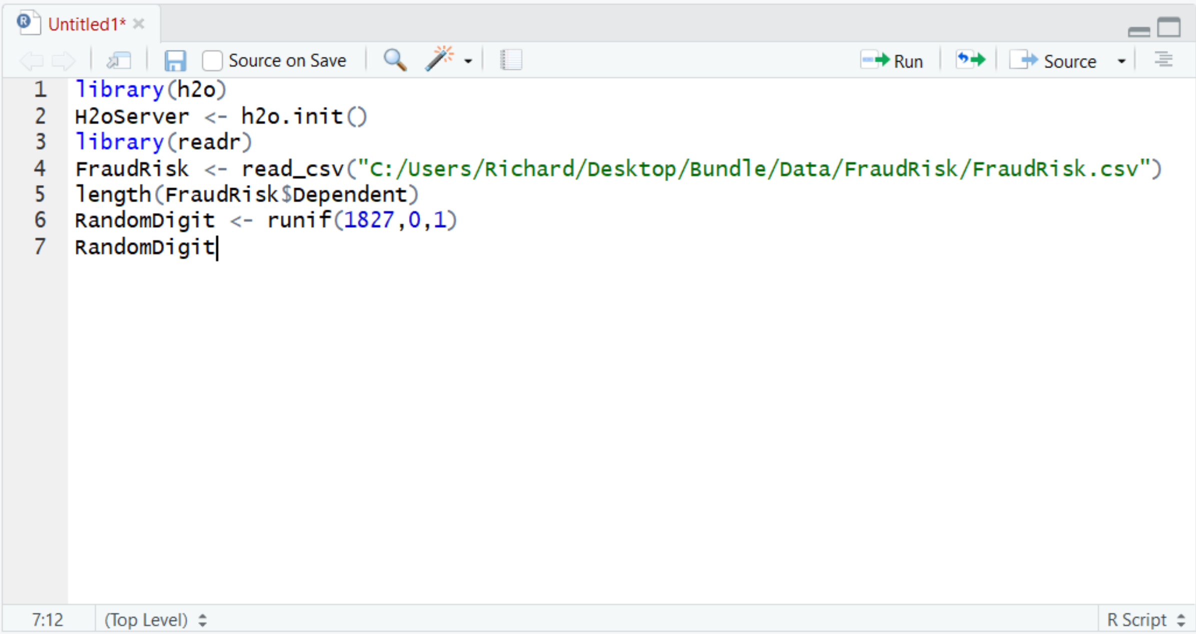
RandomDigit <- runif(1827,0,1)

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
```

Run the line of script to console:

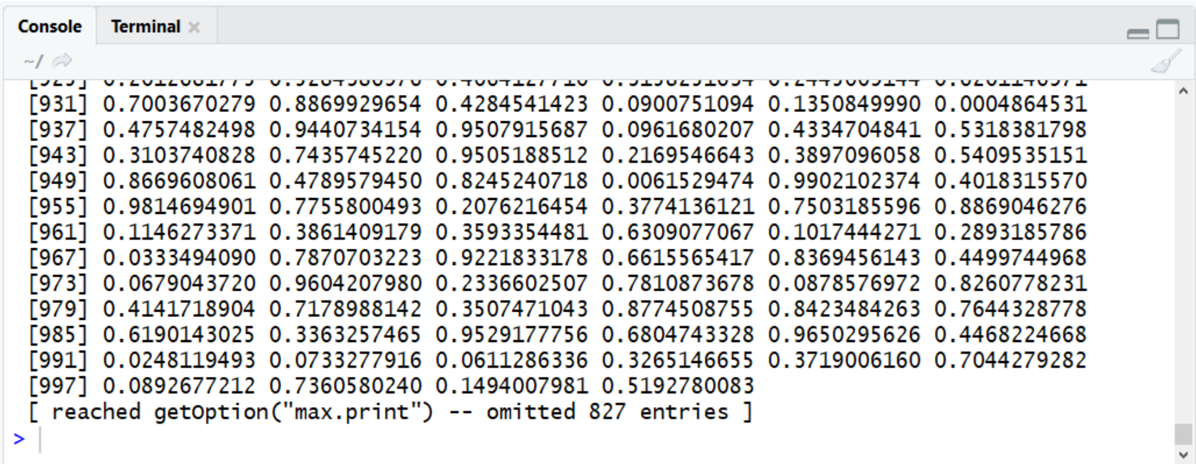
```
> FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
Parsed with column specification:
cols(
  .default = col_integer(),
  Type = col_character(),
  Transaction_Amt = col_double(),
  Sum_Transactions_1_Day = col_double(),
  Sum_ATM_Transactions_1_Day = col_double()
)
See spec(...) for full column specifications.
> length(FraudRisk$Dependent)
[1] 1827
> RandomDigit <- runif(1827,0,1)
>
```

A vector containing random digits, of same length as the dataframe, has been created. Validate vector by typing:



```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
```

Run the line of script to console:



```
[931] 0.7003670279 0.8869929654 0.4284541423 0.0900751094 0.1350849990 0.0004864531
[937] 0.4757482498 0.9440734154 0.9507915687 0.0961680207 0.4334704841 0.5318381798
[943] 0.3103740828 0.7435745220 0.9505188512 0.2169546643 0.3897096058 0.5409535151
[949] 0.8669608061 0.4789579450 0.8245240718 0.0061529474 0.9902102374 0.4018315570
[955] 0.9814694901 0.7755800493 0.2076216454 0.3774136121 0.7503185596 0.8869046276
[961] 0.1146273371 0.3861409179 0.3593354481 0.6309077067 0.1017444271 0.2893185786
[967] 0.0333494090 0.7870703223 0.9221833178 0.6615565417 0.8369456143 0.4499744968
[973] 0.0679043720 0.9604207980 0.2336602507 0.7810873678 0.0878576972 0.8260778231
[979] 0.4141718904 0.7178988142 0.3507471043 0.8774508755 0.8423484263 0.7644328778
[985] 0.6190143025 0.3363257465 0.9529177756 0.6804743328 0.9650295626 0.4468224668
[991] 0.0248119493 0.0733277916 0.0611286336 0.3265146655 0.3719006160 0.7044279282
[997] 0.0892677212 0.7360580240 0.1494007981 0.5192780083
[ reached getOption("max.print") -- omitted 827 entries ]
>
```

The random digits are written out showing there to be values created, on a random basis, between 0 and 1 with a high degree of precision. Append this vector to the dataframe as using Dplyr and Mutate:

library(dplyr)

FraudRisk <- mutate(FraudRisk,RandomDigit)

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
```

Run the block of script to console:

```
 [943] 0.1606812305 0.8045387901 0.2077633315 0.4320637623 0.6611052291 0.8583060394
 [949] 0.8209645993 0.2940145507 0.1427181836 0.2581601918 0.0886390754 0.9648075826
 [955] 0.8168699113 0.8946601977 0.5517767901 0.8982275210 0.9222396063 0.0811792270
 [961] 0.3106482869 0.3009677513 0.3248887684 0.5904285000 0.7647507291 0.9241362538
 [967] 0.4212970622 0.9807689115 0.3459127636 0.6565544992 0.9542271199 0.2306949482
 [973] 0.3711250226 0.8636053961 0.8408031585 0.2814268139 0.1668819406 0.1150107309
 [979] 0.6165395183 0.1532795408 0.4579111221 0.8064513393 0.1472172076 0.9171782045
 [985] 0.6820984862 0.2851745649 0.0613210641 0.1101909750 0.9304570444 0.2080441082
 [991] 0.0798152182 0.5042636776 0.3033120122 0.8976760195 0.6738648931 0.7304938701
 [997] 0.1080167792 0.1600905121 0.1830108017 0.1732002040
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
>
```

The RandomDigit vector is now appended to the FraudRisk dataframe and can be used in sub setting and splitting. Create the cross-validation dataset by creating a filter creating a new data frame by assignment:

CV <- filter(FraudRisk,RandomDigit < 0.2)

```r
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
```

Run the line of script to console:

```
 [949] 5.759238e-01 5.862066e-01 9.616989e-01 8.200731e-01 2.735855e-01 7.786797e-01
 [955] 5.474996e-01 5.380250e-01 9.095031e-01 9.296226e-01 3.774134e-01 7.933756e-01
 [961] 6.251487e-01 8.568623e-01 3.984767e-01 3.083549e-01 5.020710e-01 3.198431e-01
 [967] 4.000619e-01 9.162673e-01 8.207646e-01 2.616130e-01 2.510059e-01 5.201643e-01
 [973] 6.293230e-01 9.669942e-01 8.669715e-01 3.460082e-01 4.841085e-01 6.675613e-01
 [979] 1.176654e-01 8.889983e-01 3.502810e-01 6.497021e-01 8.874766e-03 5.154247e-01
 [985] 5.104639e-01 4.714951e-01 5.567964e-01 3.729349e-01 9.287575e-01 4.712628e-01
 [991] 1.829695e-01 5.655082e-01 9.593853e-01 6.147644e-01 8.168282e-01 9.048224e-01
 [997] 5.320798e-01 3.175046e-01 1.081090e-01 7.836792e-01
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
>
```

A new data frame by the name of CV has been created. Observe the CV data frame length:

length(CV$Dependent)

```r
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
```

Run the line of script to console:

```
 [982] 0.902189568 0.229194473 0.748027796
 [985] 0.547615160 0.517380027 0.418006948
 [988] 0.774244435 0.692429334 0.088466370
 [991] 0.498228362 0.971793567 0.949895006
 [994] 0.404990785 0.122788988 0.571297422
 [997] 0.744410862 0.397442776 0.023933484
[1000] 0.408841250
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
> length(CV$Dependent)
[1] 364
>
```

It can be seen that the data frame has 386 records, which is broadly 20% of the FraudRisk data frames records. The task remains to create the training dataset, which is similar albeit sub setting for a larger opposing random digit filter:

Training <- filter(FraudRisk,RandomDigit >= 0.2)

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
```

Run the line of script to console:

```
 [985] 0.547615160 0.517380027 0.418006948
 [988] 0.774244435 0.692429334 0.088466370
 [991] 0.498228362 0.971793567 0.949895006
 [994] 0.404990785 0.122788988 0.571297422
 [997] 0.744410862 0.397442776 0.023933484
[1000] 0.408841250
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
> length(CV$Dependent)
[1] 364
> Training <- filter(FraudRisk,RandomDigit >= 0.2)
>
```

Validate the length of the Training data frame:

length(Training$Dependent)

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
```

Run the line of script to console:

```
 [991] 0.498228362 0.971793567 0.949895006
 [994] 0.404990785 0.122788988 0.571297422
 [997] 0.744410862 0.397442776 0.023933484
[1000] 0.408841250
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
> length(CV$Dependent)
[1] 364
> Training <- filter(FraudRisk,RandomDigit >= 0.2)
> length(Training$Dependent)
[1] 1463
>
```

It can be observed that the Training dataset is 1463 records in length, which is broadly 70% of the file. So not to accidently use the RandomDigit vector in training, drop it from the Training and CV data frames:

CV$RandomDigit <- NULL

Training$RandomDigit <- NULL

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
```

Run the block of script to console:

```
 [997] 0.744410862 0.397442776 0.023933484
[1000] 0.408841250
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
> length(CV$Dependent)
[1] 364
> Training <- filter(FraudRisk,RandomDigit >= 0.2)
> length(Training$Dependent)
[1] 1463
> CV$RandomDigit <- NULL
> Training$RandomDigit <- NULL
>
```

H2O requires that the Dependent Variable is a factor, it is after all a classification problem. Convert the dependent variable to a factor for the training and cross validation dataset:

```
library(h2o)
H2oServer <- h2o.init()
library(readr)
FraudRisk <- read_csv("C:/Users/Richard/Desktop/Bundle/Data/FraudRisk/FraudRisk.csv")
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
CV$Dependent <- factor(CV$Dependent)
Training$Dependent <- factor(Training$Dependent)
```

Run the line of script to console:

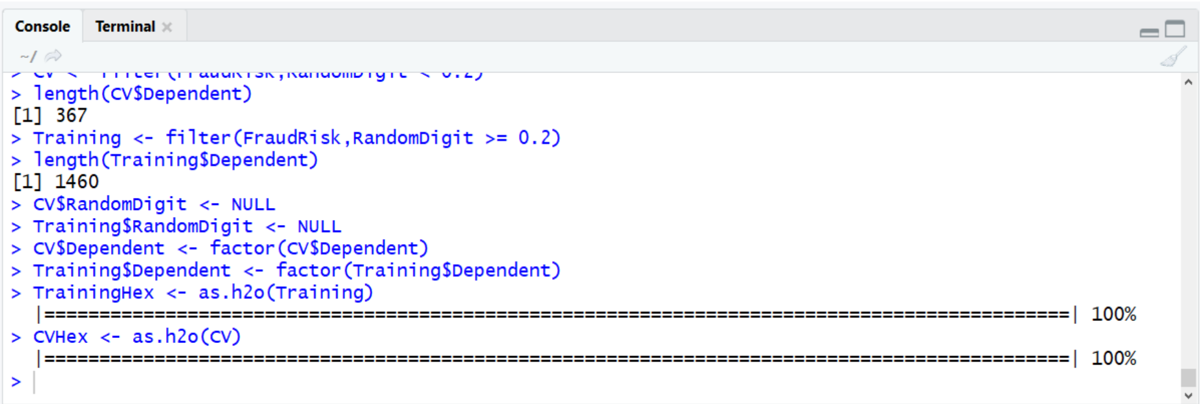
```
 [ reached getOption("max.print") -- omitted 827 entries ]
> library(dplyr)
> FraudRisk <- mutate(FraudRisk,RandomDigit)
> CV <- filter(FraudRisk,RandomDigit < 0.2)
> length(CV$Dependent)
[1] 364
> Training <- filter(FraudRisk,RandomDigit >= 0.2)
> length(Training$Dependent)
[1] 1463
> CV$RandomDigit <- NULL
> Training$RandomDigit <- NULL
> CV$Dependent <- factor(CV$Dependent)
> Training$Dependent <- factor(Training$Dependent)
>
```

At this stage, there now exists a randomly selected Training dataset as well as a randomly selection Cross Validation training set. Keep in mind that H2O requires that the dataframe is converted to the native hex format, achieved through the creation of a parsed data object for each dataset. Think of this process as being the loading of data into the H2O server, more so than a conversion to Hex:

Training.hex <- as.h2o(Training)

CV.hex <- as.h2o(CV)

Run the block of script to console:



All models that are available to be trained via the Flow interface are available via the R interface, with the hex files being ready to be passed as parameters.

Procedure 6: Creating a Neural Network with R

Although all of the work is offloaded to H2O, the instruction to train a model looks a lot like previous examples where a variety of R packages have been used. In this example the deeplearning function of the H2O package is going to be used (this is really the only reason that we are using H2O in the first place).

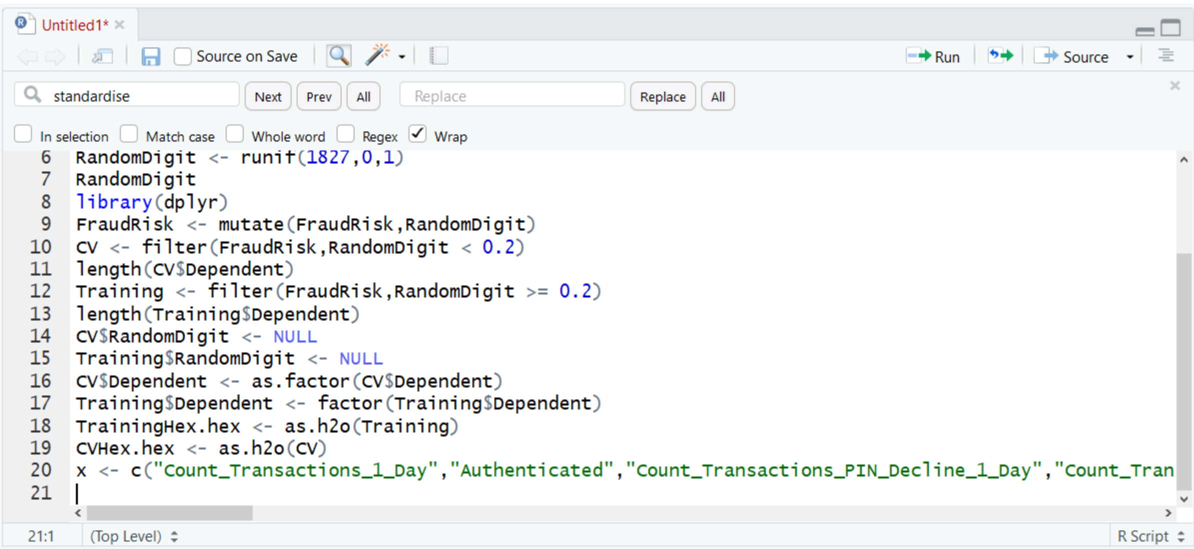
In order to make the command easier to understand, typed parameters will be used as follows:

Parameter	Description
x	c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Transactions_Declined_1_Day","Count_Unsafe_Terminals_1_Day","Count_In_Person_1_Day","Count_Internet_1_Day","ATM","Count_ATM_1_Day","Count_Over_30_SEK_1_Day","In_Person","Transaction_Amt","Sum_Transactions_1_Day","Sum_ATM_Transactions_1_Day","Foreign","Different_Country_Transactions_1_Week","Different_Merchant_Types_1_Week","Diffe

	rent_Decline_Reasons_1_Day","Different_Cities_1_Week","Count_Same_Merchant_Used_Before_1_Week","Has_Been_Abroad","Cash_Transaction","High_Risk_Country")
y	c("Dependent")
training_frame	TrainingHex
validation_frame	CVHex
standardise	FALSE
activation	Rectifier
epochs	50
seed	12345
hidden	5
variable_importance	TRUE
nfolds	5
adaptive_rate	FALSE

The deeplearning function in H2O takes a function two vectors that contain the dependent and independent variables. For readability, create these string vectors to be passed to the deeplearning function in advance, rather than use the c() function, inside the function call. To create a list of eligible independent variables for the purposes of this example, enter:

x <-
c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Transactions_Declined_1_Day","Count_Unsafe_Terminals_1_Day","Count_In_Person_1_Day","Count_Internet_1_Day","ATM","Count_ATM_1_Day","Count_Over_30_SEK_1_Day","In_Person","Transaction_Amt","Sum_Transactions_1_Day","Sum_ATM_Transactions_1_Day","Foreign","Different_Country_Transactions_1_Week","Different_Merchant_Types_1_Week","Different_Decline_Reasons_1_Day","Different_Cities_1_Week","Count_Same_Merchant_Used_Before_1_Week","Has_Been_Abroad","Cash_Transaction","High_Risk_Country")



Run the line of script to console:

```
Console   Terminal ×
~/
> CV$RandomDigit <- NULL
> Training$RandomDigit <- NULL
> CV$Dependent <- as.factor(CV$Dependent)
> Training$Dependent <- factor(Training$Dependent)
> TrainingHex.hex <- as.h2o(Training)
  |=============================================================================| 100%
> CVHex.hex <- as.h2o(CV)
  |=============================================================================| 100%
> x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Transactions_Declined_1_Day","Count_Unsafe_Terminals_1_Day","Count_In_Person_1_Day","Count_Internet_1_Day","ATM","Count_ATM_1_Day","Count_Over_30_SEK_1_Day","In_Person","Transaction_Amt","Sum_Transactions_1_Day","Sum_ATM_Transactions_1_Day","Foreign","Different_Country_Transactions_1_Week","Different_Merchant_Types_1_Week","Different_Decline_Reasons_1_Day","Different_Cities_1_Week","Count_Same_Merchant_Used_Before_1_Week","Has_Been_Abroad","Cash_Transaction","High_Risk_Country")
> 
```

To instruct H2O to begin deep learning, enter:

Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.hex,activation="Rectifier", epochs=50,seed=12345,hidden=5,variable_importance=TRUE,nfolds=5,adaptive_rate=FALSE,standardize=TRUE)

```
Untitled1* ×
Source on Save                                    Run    Source
standardise     Next  Prev  All    Replace    Replace  All
In selection  Match case  Whole word  Regex  Wrap
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
CV$Dependent <- as.factor(CV$Dependent)
Training$Dependent <- factor(Training$Dependent)
TrainingHex.hex <- as.h2o(Training)
CVHex.hex <- as.h2o(CV)
x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Tran
Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.h
21:228   (Top Level)                                                              R Script
```

Run the line of script to console:

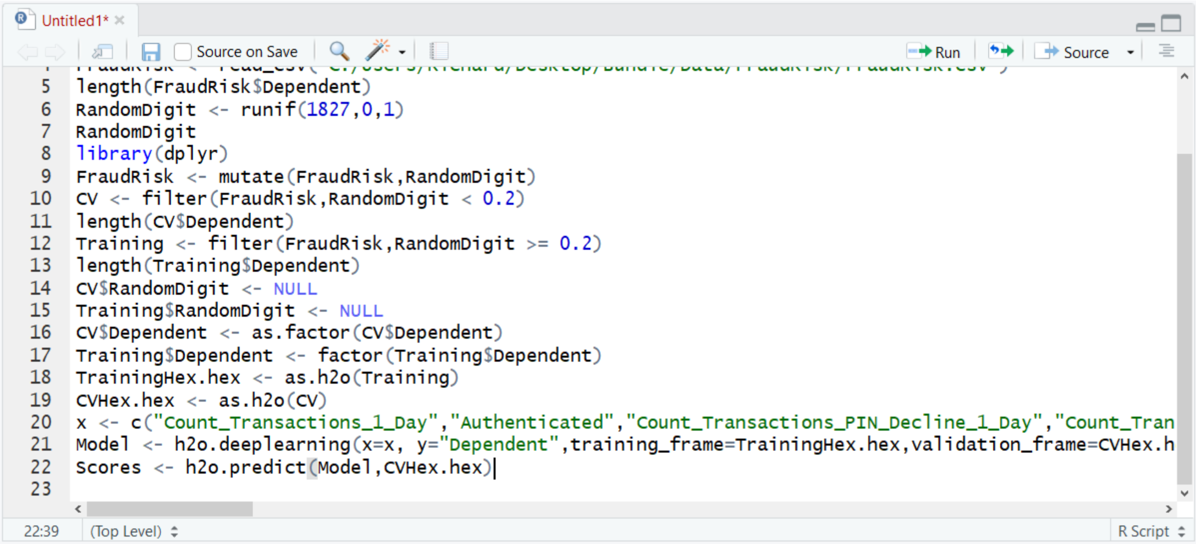
```
Console   Terminal ×
~/
> TrainingHex.hex <- as.h2o(Training)
  |=============================================================================| 100%
> CVHex.hex <- as.h2o(CV)
  |=============================================================================| 100%
> x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Transactions_Declined_1_Day","Count_Unsafe_Terminals_1_Day","Count_In_Person_1_Day","Count_Internet_1_Day","ATM","Count_ATM_1_Day","Count_Over_30_SEK_1_Day","In_Person","Transaction_Amt","Sum_Transactions_1_Day","Sum_ATM_Transactions_1_Day","Foreign","Different_Country_Transactions_1_Week","Different_Merchant_Types_1_Week","Different_Decline_Reasons_1_Day","Different_Cities_1_Week","Count_Same_Merchant_Used_Before_1_Week","Has_Been_Abroad","Cash_Transaction","High_Risk_Country")
> Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.hex,activation="Rectifier",epochs=50,seed=12345,hidden=5,variable_importance=TRUE,nfolds=5,adaptive_rate=FALSE,standardize=TRUE)
  |=============================================================================| 100%
> 
```

Feedback from the H2O cluster will be received, detailing training progress.

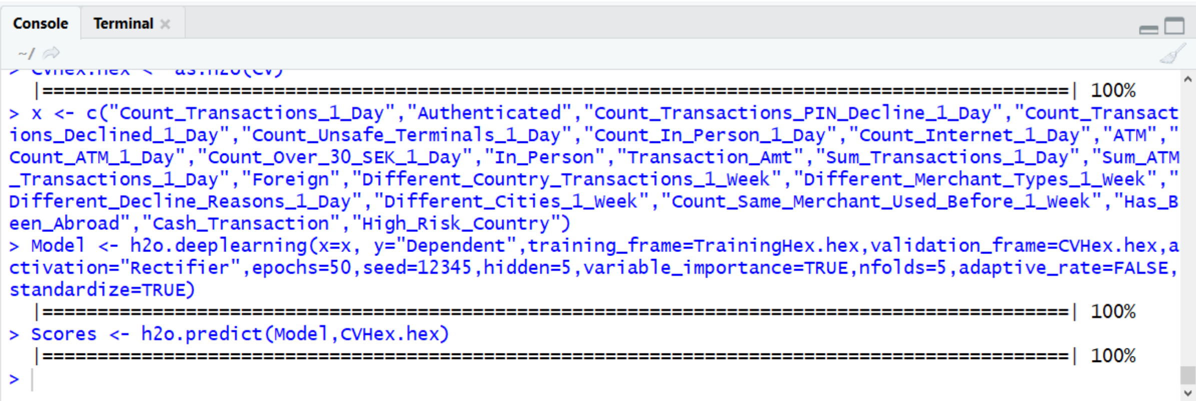
Procedure 7: Recalling a Neural Network with R

Once a model is trained in H2O it can be recalled very gracefully with the predict() function of the H2O package. It is a simple matter of passing the trained model and the hex dataframe to be used for recall:

Scores <- h2o.predict(Model,CVHex.hex)



Run the line of script to console:



A progress bar is broadcast from the H2O server and will be written out to the console. To review the output, enter the object:

Scores

```
Untitled1*
length(FraudRisk$Dependent)
RandomDigit <- runif(1827,0,1)
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
CV$Dependent <- as.factor(CV$Dependent)
Training$Dependent <- factor(Training$Dependent)
TrainingHex.hex <- as.h2o(Training)
CVHex.hex <- as.h2o(CV)
x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Tran
Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.h
Scores <- h2o.predict(Model,CVHex.hex)
Scores
```

Run the line of script to console:

```
  |======================================================================================| 100%
> Scores <- h2o.predict(Model,CVHex.hex)
  |======================================================================================| 100%
> Scores
  predict        p0        p1
1       0 0.7611897 0.2388103
2       1 0.2602582 0.7397418
3       0 0.7841905 0.2158095
4       0 0.9339192 0.0660808
5       0 0.8721822 0.1278178
6       1 0.2396435 0.7603565

[351 rows x 3 columns]
>
```

The Scores output appears similar to a matrix, but it has created a vector which details the actual prediction for a record, hence, this can be subset to a final vector detailing the predictions:

Predict <- Scores[1]

```
Untitled1*
RandomDigit
library(dplyr)
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
CV$Dependent <- as.factor(CV$Dependent)
Training$Dependent <- factor(Training$Dependent)
TrainingHex.hex <- as.h2o(Training)
CVHex.hex <- as.h2o(CV)
x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Tran
Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.h
Scores <- h2o.predict(Model,CVHex.hex)
Scores
Predict <- Scores[1]
```

Run the line of script to console:

```
Console   Terminal
~/
  |==========================================================================================| 100%
> Scores <- h2o.predict(Model,CVHex.hex)
  |==========================================================================================| 100%
> Scores
  predict         p0          p1
1       1 0.11963822 0.880361783
2       1 0.08817501 0.911824988
3       1 0.24004925 0.759950749
4       0 0.99885389 0.001146109
5       1 0.02847112 0.971528876
6       1 0.20062463 0.799375367

[386 rows x 3 columns]
> Predict <- Scores[1]
> 
```

The Predict vector can be compared to the Dependent vector of the CV dataframe in the same manner as previous models within R to obtain Confusion Matrices as well a ROC curves.

Module 16: Monte Carlo Model Simulation.

Monte Carlo Simulation is a technique to create many random simulations based upon a random case (i.e. a transaction). The random value can be forced to obey certain statistical assumptions, which in this example will be a triangular distribution. Monte Carlo simulation is an enormous topic in its own right yet these procedures are intended to give just a basic overview of the tool and allow for the simulation of models created in these procedures.

Simulation for Communication refers to being able to run models based on explainable statically assumptions so to facilitate expectation setting for the model's impact. Furthermore, that millions of random simulations will be exposed to the model, where records of both the randomly generated record and the output are retained, Monte Carlo simulation can help identify scenarios where there is potential for optimisation or risk mitigation.

There are many types of distributions that can be randomly simulated, supported by functions in R. The runif() and rnorm() functions are the most commonly used. The runif() function creates discrete values between a high and low amount. The rnorm function creates values inside a normal distribution, taking the minimum, maximum, mean and standard deviation as parameters.

For most business simulations, the triangular distribution is most practical, given that the normal distribution is quite rarely seen.

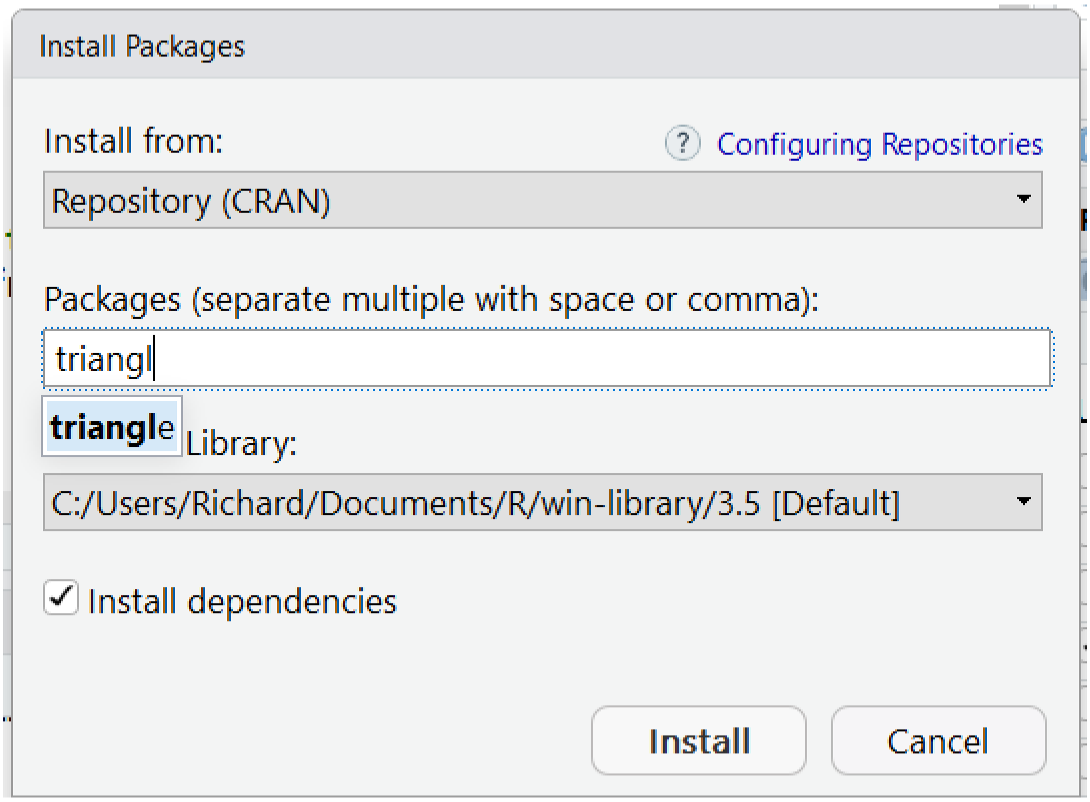
Procedure 1: Create Discrete Vectors with triangle for each model parameter

In this example, the result is the simulation of the neural network model that was created in H2O. It follows that we need to create a dataframe with the same specification the training data set.

For the purposes of our example, we are going to create triangular distributions comprised of the Minimum Value, the Maximum Value and the Mean. This simulated dataframe will be 100,000 records in length.

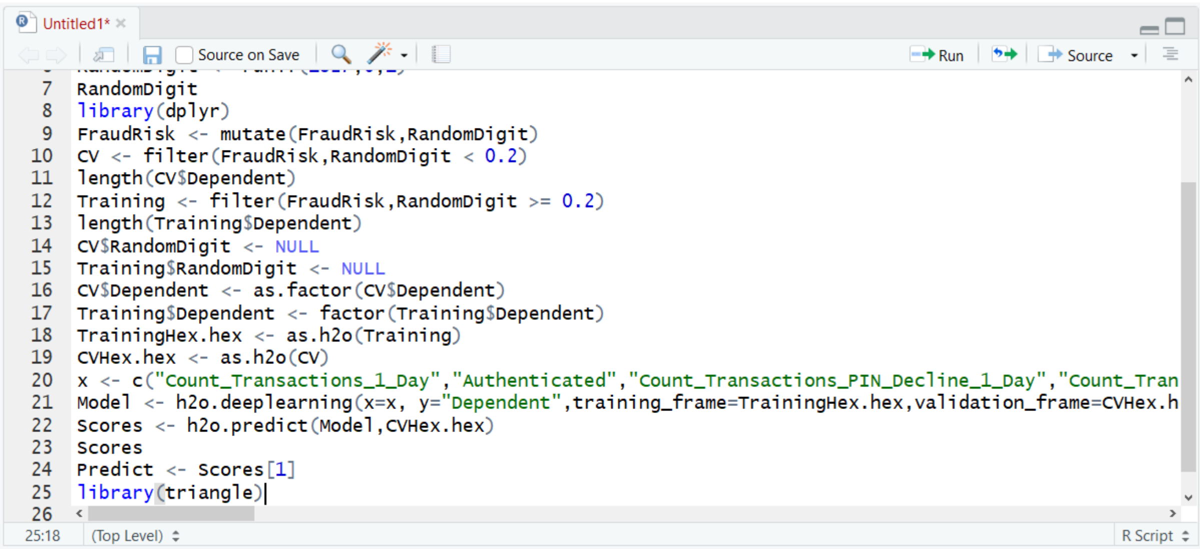
This procedure will focus on creating this vector for a single variable, before providing a block of script to achieve this for each variable at the end of the procedure.

Firstly, install the triangle package:



Load the library:

library(triangle)



Run the line of script to console:

```
> Scores <- h2o.predict(Model,cvHex.hex)
  |==========================================================================================| 100%
> Scores
  predict          p0          p1
1       0 0.87627777 0.123722232
2       1 0.06930181 0.930698188
3       1 0.13020317 0.869796834
4       0 0.99223909 0.007760913
5       1 0.06195748 0.938042525
6       1 0.10581270 0.894187298

[358 rows x 3 columns]
> Predict <- Scores[1]
> library(triangle)
> 
```

The rtriangle() function accepts four parameters:

Name	Description	Example
Simulations	This is the size of the return vector and number of simulations to create.	100000
Min	The smallest value to be created in the simulation.	0
Max	The largest value to be created in the simulation.	100
Mean or Mode	The Mean or Mode used to skew the distribution to more closely align to the real data.	10

The dataframe needs to be as closely aligned to the real data as possible and as such the triangular distribution points are going to be taken from the training dataframe rather than created manually. To create a vector for the first variable used in H2O model training use the following line of script:

Count_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Count_Transactions_1_Day),max(FraudRisk$Count_Transactions_1_Day),mean(FraudRisk$Count_Transactions_1_Day))

```
ons_PIN_Decline_1_Day","Count_Transactions_Declined_1_Day","Count_Unsafe_Terminals_1_Day","Count_In_P
ngHex.hex,validation_frame=CVHex.hex,activation="Rectifier",epochs=50,seed=12345,hidden=c(5,5,5),vari




ransactions_1_Day),max(FraudRisk$Count_Transactions_1_Day),mean(FraudRisk$Count_Transactions_1_Day))
```

Run the line of script to console:

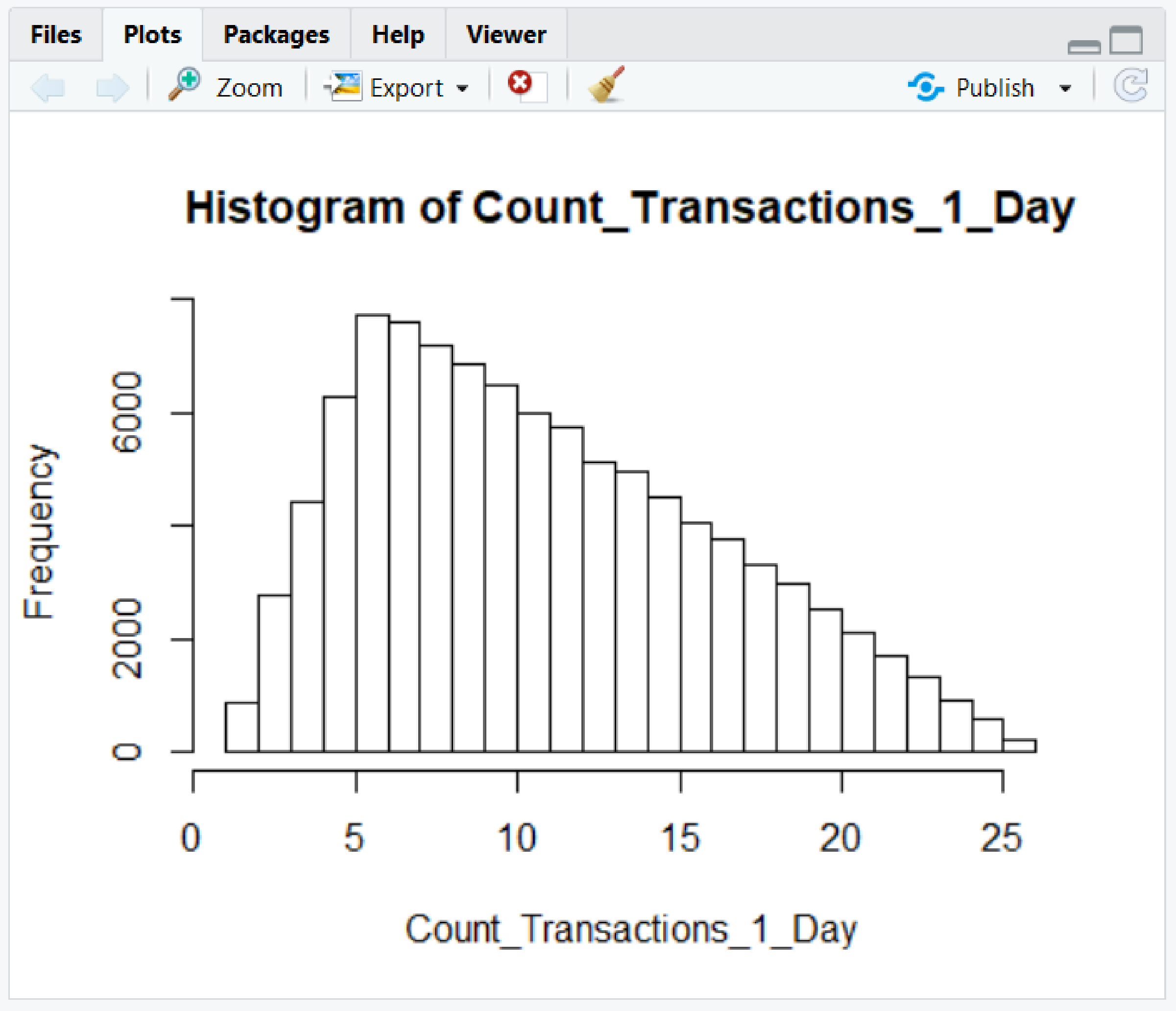
```
  predict         p0          p1
1       0 0.87627777 0.123722232
2       1 0.06930181 0.930698188
3       1 0.13020317 0.869796834
4       0 0.99223909 0.007760913
5       1 0.06195748 0.938042525
6       1 0.10581270 0.894187298

[358 rows x 3 columns]
> Predict <- Scores[1]
> library(triangle)
> Count_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Count_Transactions_1_Day),max(FraudRisk$Count
_Transactions_1_Day),mean(FraudRisk$Count_Transactions_1_Day))
>
```

Validate the vector by inspecting it as a histogram:

hist(Count_Transactions_1_Day)

```
FraudRisk <- mutate(FraudRisk,RandomDigit)
CV <- filter(FraudRisk,RandomDigit < 0.2)
length(CV$Dependent)
Training <- filter(FraudRisk,RandomDigit >= 0.2)
length(Training$Dependent)
CV$RandomDigit <- NULL
Training$RandomDigit <- NULL
CV$Dependent <- as.factor(CV$Dependent)
Training$Dependent <- factor(Training$Dependent)
TrainingHex.hex <- as.h2o(Training)
CVHex.hex <- as.h2o(CV)
x <- c("Count_Transactions_1_Day","Authenticated","Count_Transactions_PIN_Decline_1_Day","Count_Tran
Model <- h2o.deeplearning(x=x, y="Dependent",training_frame=TrainingHex.hex,validation_frame=CVHex.h
Scores <- h2o.predict(Model,CVHex.hex)
Scores
Predict <- Scores[1]
library(triangle)
Count_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Count_Transactions_1_Day),max(FraudRisk$C
hist(Count_Transactions_1_Day)
```

Run the line of script to console:



It can be seen that a triangular distribution has been created, slightly skewed to axis. The task now remains to repeat this for each of the variables required of the H2O model. The construct and principle for this procedure will be the same, for each variable:

Authenticated <- rtriangle(100000,min(FraudRisk$Authenticated),max(FraudRisk$Authenticated),mean(FraudRisk$Authenticated))

Count_Transactions_PIN_Decline_1_Day <- rtriangle(100000,min(FraudRisk$Count_Transactions_PIN_Decline_1_Day),max(FraudRisk$Count_Transactions_PIN_Decline_1_Day),mean(FraudRisk$Count_Transactions_PIN_Decline_1_Day))

Count_Transactions_Declined_1_Day <- rtriangle(100000,min(FraudRisk$Count_Transactions_Declined_1_Day),max(FraudRisk$Count_Transactions_Declined_1_Day),mean(FraudRisk$Count_Transactions_Declined_1_Day))

Count_Unsafe_Terminals_1_Day <- rtriangle(100000,min(FraudRisk$Count_Unsafe_Terminals_1_Day),max(FraudRisk$Count_Unsafe_Terminals_1_Day),mean(FraudRisk$Count_Unsafe_Terminals_1_Day))

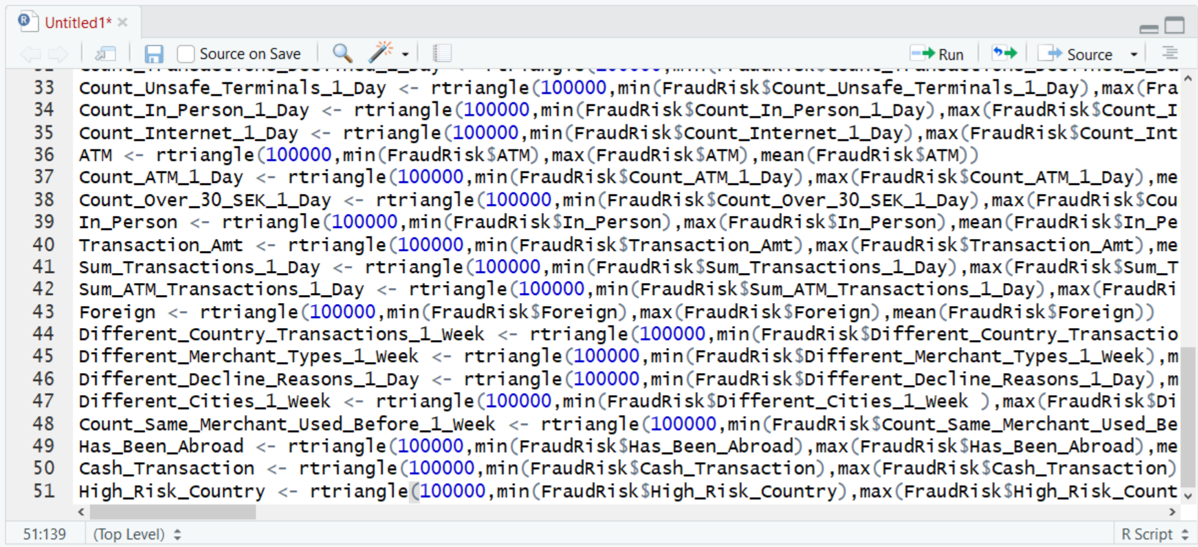
Count_In_Person_1_Day <- rtriangle(100000,min(FraudRisk$Count_In_Person_1_Day),max(FraudRisk$Count_In_Person_1_Day),mean(FraudRisk$Count_In_Person_1_Day))

```
Count_Internet_1_Day <- rtriangle(100000,min(FraudRisk$Count_Internet_1_Day),max(FraudRisk$Count_Internet_1_Day),mean(FraudRisk$Count_Internet_1_Day))

ATM <- rtriangle(100000,min(FraudRisk$ATM),max(FraudRisk$ATM),mean(FraudRisk$ATM))

Count_ATM_1_Day <- rtriangle(100000,min(FraudRisk$Count_ATM_1_Day),max(FraudRisk$Count_ATM_1_Day),mean(FraudRisk$Count_ATM_1_Day))

Count_Over_30_SEK_1_Day <- rtriangle(100000,min(FraudRisk$Count_Over_30_SEK_1_Day),max(FraudRisk$Count_Over_30_SEK_1_Day),mean(FraudRisk$Count_Over_30_SEK_1_Day))

In_Person <- rtriangle(100000,min(FraudRisk$In_Person),max(FraudRisk$In_Person),mean(FraudRisk$In_Person))

Transaction_Amt <- rtriangle(100000,min(FraudRisk$Transaction_Amt),max(FraudRisk$Transaction_Amt),mean(FraudRisk$Transaction_Amt))

Sum_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_Transactions_1_Day),max(FraudRisk$Sum_Transactions_1_Day),mean(FraudRisk$Sum_Transactions_1_Day))

Sum_ATM_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_ATM_Transactions_1_Day),max(FraudRisk$Sum_ATM_Transactions_1_Day),mean(FraudRisk$Sum_ATM_Transactions_1_Day))

Foreign <- rtriangle(100000,min(FraudRisk$Foreign),max(FraudRisk$Foreign),mean(FraudRisk$Foreign))

Different_Country_Transactions_1_Week <- rtriangle(100000,min(FraudRisk$Different_Country_Transactions_1_Week),max(FraudRisk$Different_Country_Transactions_1_Week),mean(FraudRisk$Different_Country_Transactions_1_Week))

Different_Merchant_Types_1_Week <- rtriangle(100000,min(FraudRisk$Different_Merchant_Types_1_Week),max(FraudRisk$Different_Merchant_Types_1_Week),mean(FraudRisk$Different_Merchant_Types_1_Week))

Different_Decline_Reasons_1_Day <- rtriangle(100000,min(FraudRisk$Different_Decline_Reasons_1_Day),max(FraudRisk$Different_Decline_Reasons_1_Day),mean(FraudRisk$Different_Decline_Reasons_1_Day))

Different_Cities_1_Week <- rtriangle(100000,min(FraudRisk$Different_Cities_1_Week ),max(FraudRisk$Different_Cities_1_Week ),mean(FraudRisk$Different_Cities_1_Week ))

Count_Same_Merchant_Used_Before_1_Week <- rtriangle(100000,min(FraudRisk$Count_Same_Merchant_Used_Before_1_Week),max(FraudRisk$Count_Same_Merchant_Used_Before_1_Week),mean(FraudRisk$Count_Same_Merchant_Used_Before_1_Week))
```

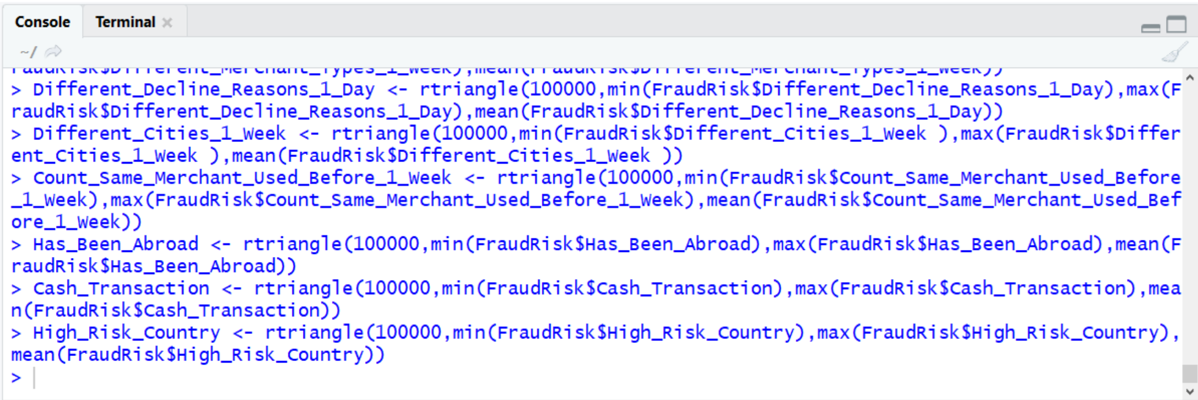
Has_Been_Abroad <- rtriangle(100000,min(FraudRisk$Has_Been_Abroad),max(FraudRisk$Has_Been_Abroad),mean(FraudRisk$Has_Been_Abroad))

Cash_Transaction <- rtriangle(100000,min(FraudRisk$Cash_Transaction),max(FraudRisk$Cash_Transaction),mean(FraudRisk$Cash_Transaction))

High_Risk_Country <- rtriangle(100000,min(FraudRisk$High_Risk_Country),max(FraudRisk$High_Risk_Country),mean(FraudRisk$High_Risk_Country))

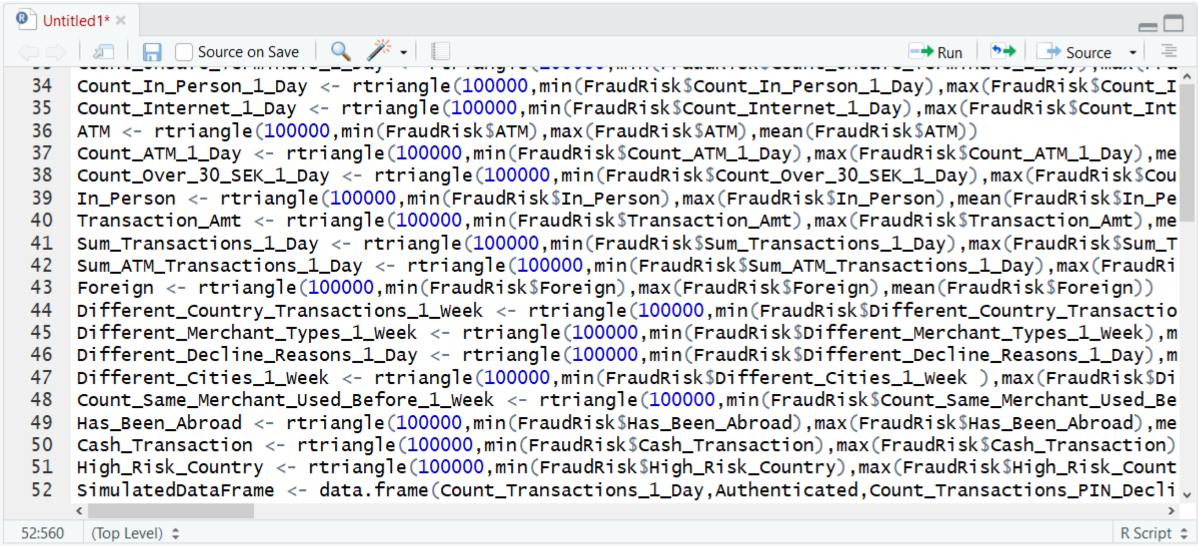


Run the block of script to console:

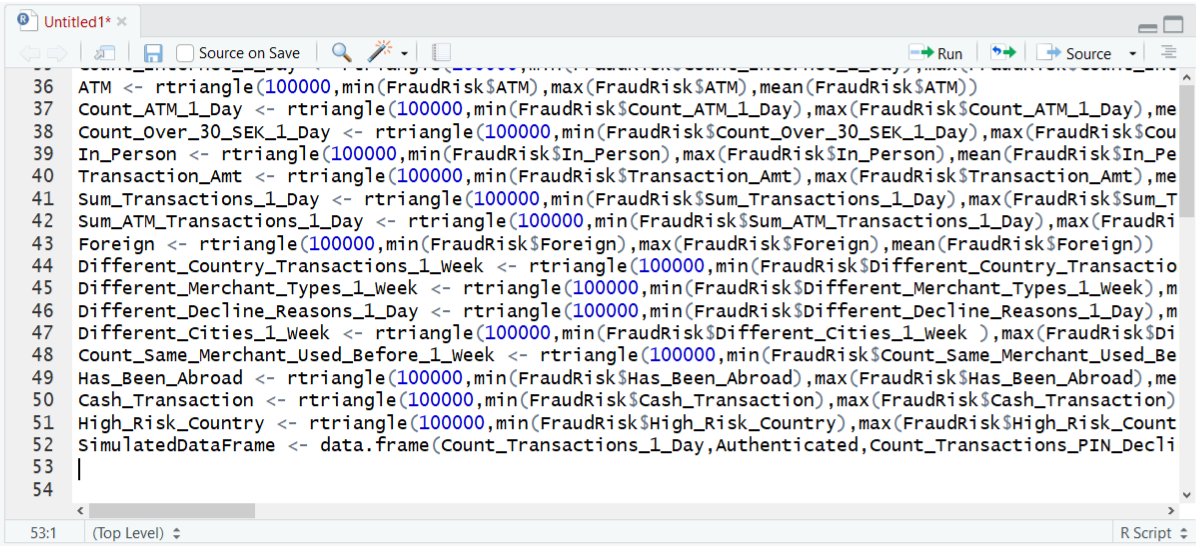


There now exists many randomly simulated vectors, created using a triangular distribution for each input variable for the H2O neural network model. They now need to be brought together in a dataframe using the data.frame function:

SimulatedDataFrame <- data.frame(Count_Transactions_1_Day,Authenticated,Count_Transactions_PIN_Decline_1_Day,Count_Transactions_Declined_1_Day,Count_Unsafe_Terminals_1_Day,Count_In_Person_1_Day,Count_Internet_1_Day,ATM,Count_ATM_1_Day,Count_Over_30_SEK_1_Day,In_Person,Transaction_Amt,Sum_Transactions_1_Day,Sum_ATM_Transactions_1_Day,Foreign,Different_Country_Transactions_1_Week,Different_Merchant_Types_1_Week,Different_Decline_Reasons_1_Day,Different_Cities_1_W eek,Count_Same_Merchant_Used_Before_1_Week,Has_Been_Abroad,Cash_Transaction,High_Risk_Country)

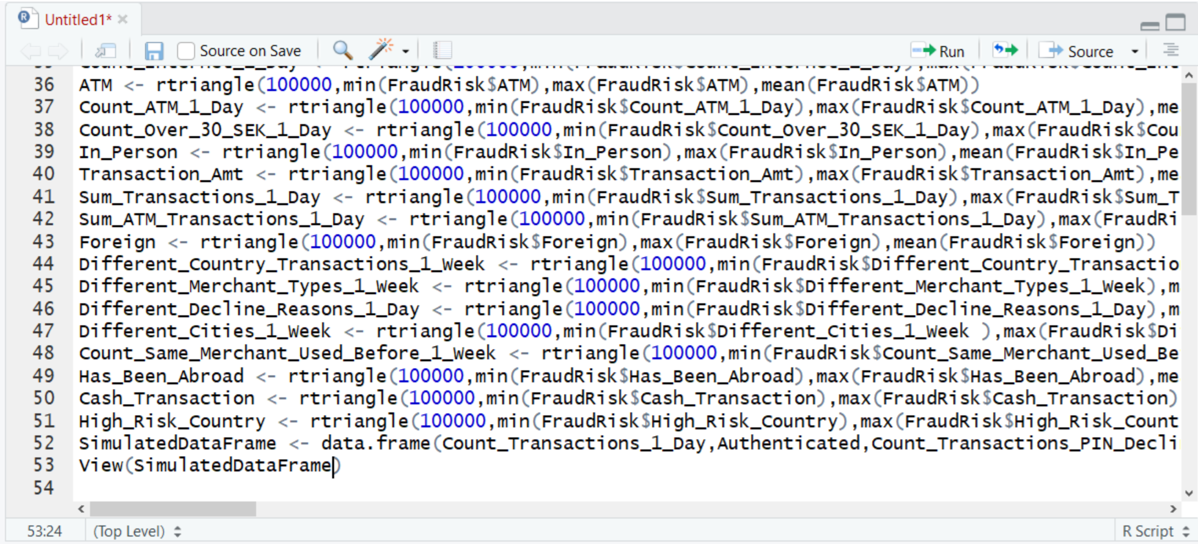


Run the line of script to console:



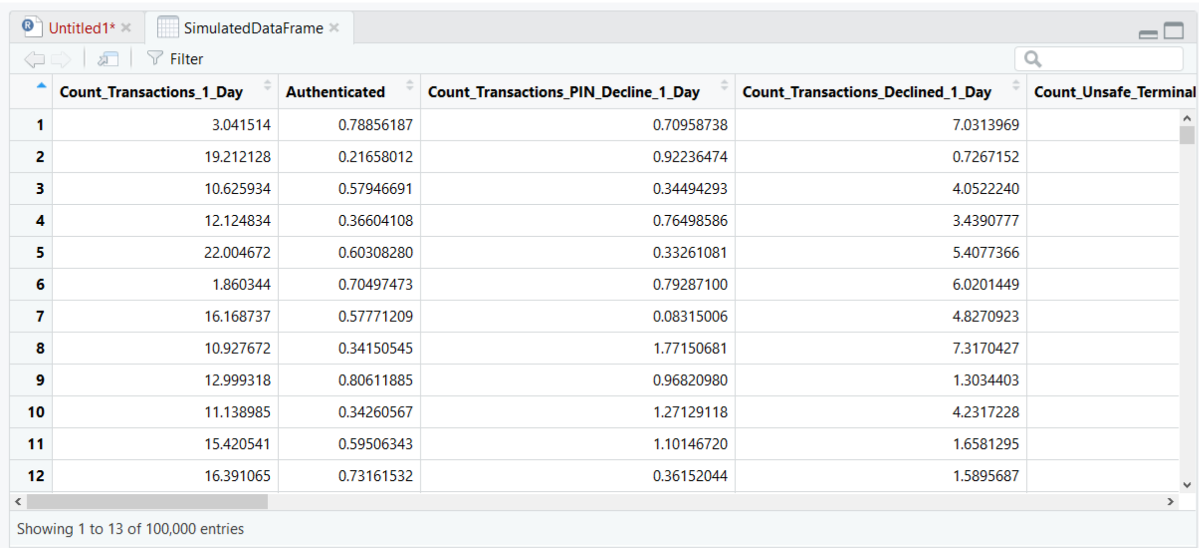
On viewing the SimuatedDataFrame, it can be seen that a new data frame has been created comprising random values. This data frame can now be used in model recall in a variety of R models:

View(SimuatedDataFrame)



```
ATM <- rtriangle(100000,min(FraudRisk$ATM),max(FraudRisk$ATM),mean(FraudRisk$ATM))
Count_ATM_1_Day <- rtriangle(100000,min(FraudRisk$Count_ATM_1_Day),max(FraudRisk$Count_ATM_1_Day),me
Count_Over_30_SEK_1_Day <- rtriangle(100000,min(FraudRisk$Count_Over_30_SEK_1_Day),max(FraudRisk$Cou
In_Person <- rtriangle(100000,min(FraudRisk$In_Person),max(FraudRisk$In_Person),mean(FraudRisk$In_Pe
Transaction_Amt <- rtriangle(100000,min(FraudRisk$Transaction_Amt),max(FraudRisk$Transaction_Amt),me
Sum_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_Transactions_1_Day),max(FraudRisk$Sum_T
Sum_ATM_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_ATM_Transactions_1_Day),max(FraudRi
Foreign <- rtriangle(100000,min(FraudRisk$Foreign),max(FraudRisk$Foreign),mean(FraudRisk$Foreign))
Different_Country_Transactions_1_Week <- rtriangle(100000,min(FraudRisk$Different_Country_Transactio
Different_Merchant_Types_1_Week <- rtriangle(100000,min(FraudRisk$Different_Merchant_Types_1_Week),m
Different_Decline_Reasons_1_Day <- rtriangle(100000,min(FraudRisk$Different_Decline_Reasons_1_Day),m
Different_Cities_1_Week <- rtriangle(100000,min(FraudRisk$Different_Cities_1_Week ),max(FraudRisk$Di
Count_Same_Merchant_Used_Before_1_Week <- rtriangle(100000,min(FraudRisk$Count_Same_Merchant_Used_Be
Has_Been_Abroad <- rtriangle(100000,min(FraudRisk$Has_Been_Abroad),max(FraudRisk$Has_Been_Abroad),me
Cash_Transaction <- rtriangle(100000,min(FraudRisk$Cash_Transaction),max(FraudRisk$Cash_Transaction)
High_Risk_Country <- rtriangle(100000,min(FraudRisk$High_Risk_Country),max(FraudRisk$High_Risk_Count
SimulatedDataFrame <- data.frame(Count_Transactions_1_Day,Authenticated,Count_Transactions_PIN_Decli
View(SimulatedDataFrame)
```

Run the line of script to console:



	Count_Transactions_1_Day	Authenticated	Count_Transactions_PIN_Decline_1_Day	Count_Transactions_Declined_1_Day	Count_Unsafe_Terminal
1	3.041514	0.78856187	0.70958738	7.0313969	
2	19.212128	0.21658012	0.92236474	0.7267152	
3	10.625934	0.57946691	0.34494293	4.0522240	
4	12.124834	0.36604108	0.76498586	3.4390777	
5	22.004672	0.60308280	0.33261081	5.4077366	
6	1.860344	0.70497473	0.79287100	6.0201449	
7	16.168737	0.57771209	0.08315006	4.8270923	
8	10.927672	0.34150545	1.77150681	7.3170427	
9	12.999318	0.80611885	0.96820980	1.3034403	
10	11.138985	0.34260567	1.27129118	4.2317228	
11	15.420541	0.59506343	1.10146720	1.6581295	
12	16.391065	0.73161532	0.36152044	1.5895687	

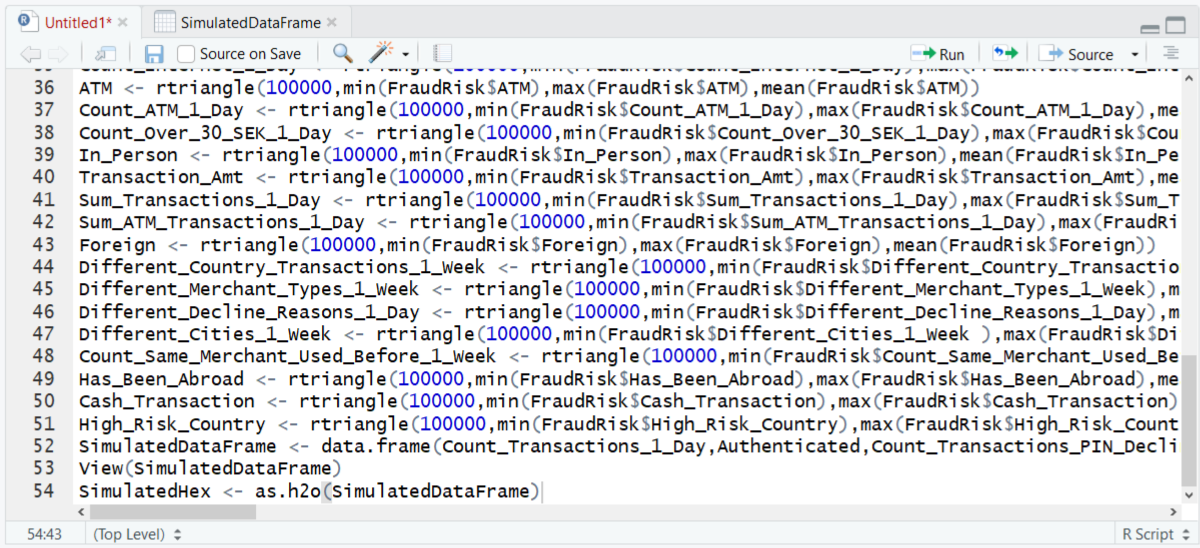
Showing 1 to 13 of 100,000 entries

Procedure 2: Process Random Data Frame against Neural Network Model

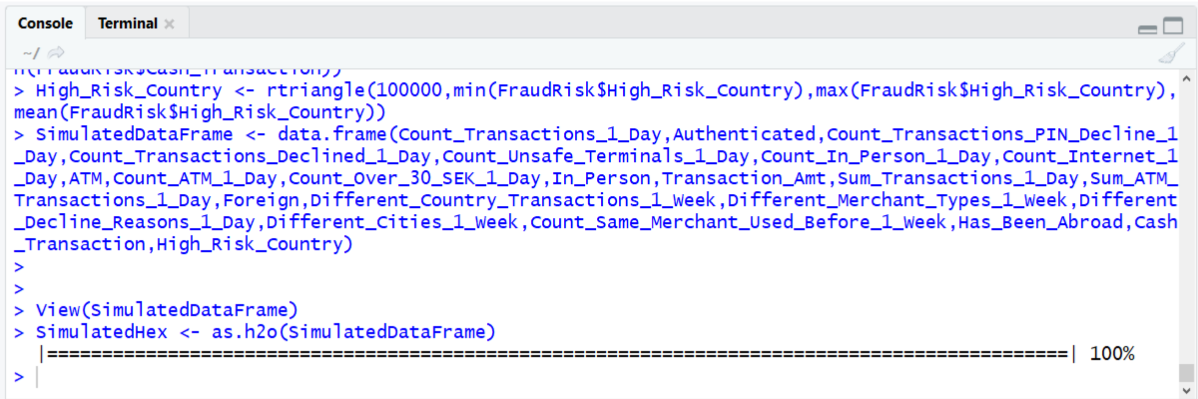
The data frame can be used with all of the machine learning algorithms presented in this guide thus far, although to use the data frame with H2O, it needs to be loaded into H2O as hex:

To load the data frame into H2O use:

SimulatedHex <- as.h2o(SimulatedDataFrame)

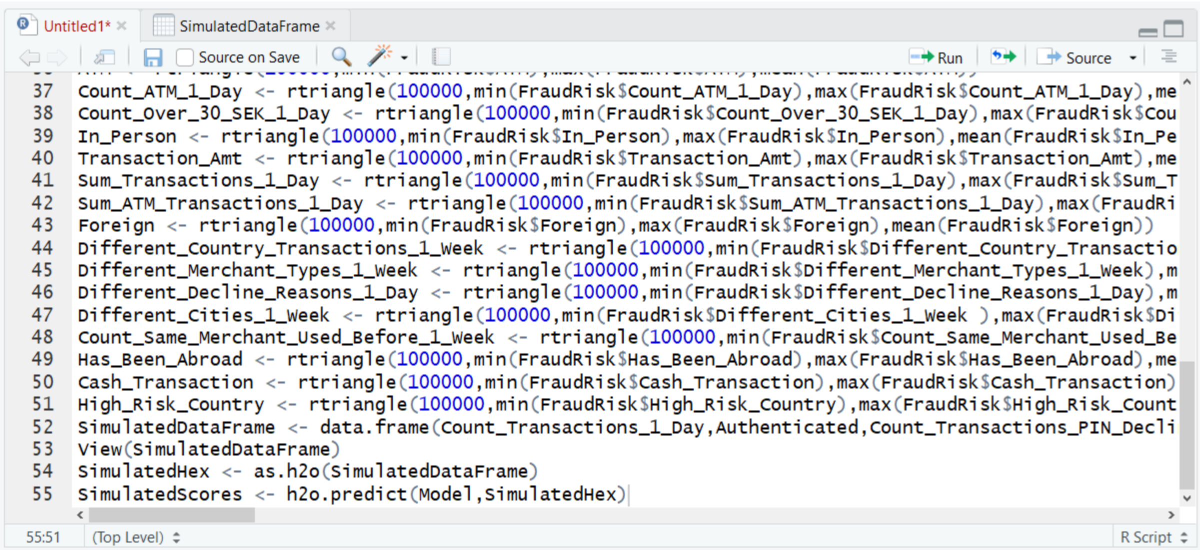


Run the line of script to console:



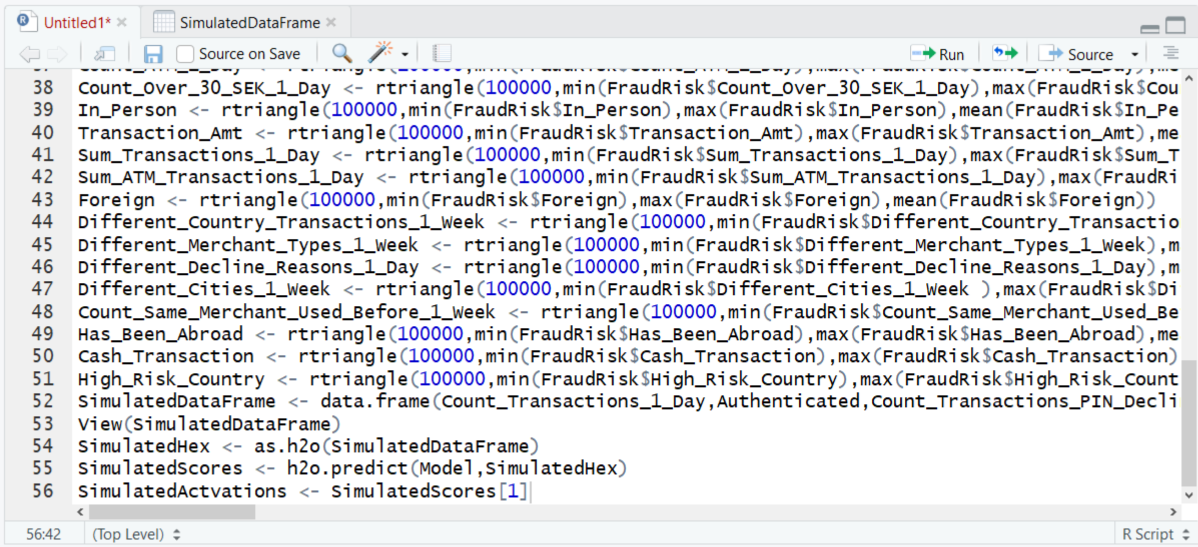
As before, use the H2O predict function to execute the model, passing the simulated dataframe in the place of real data:

SimulatedScores <- h2o.predict(Model,SimulatedHex)

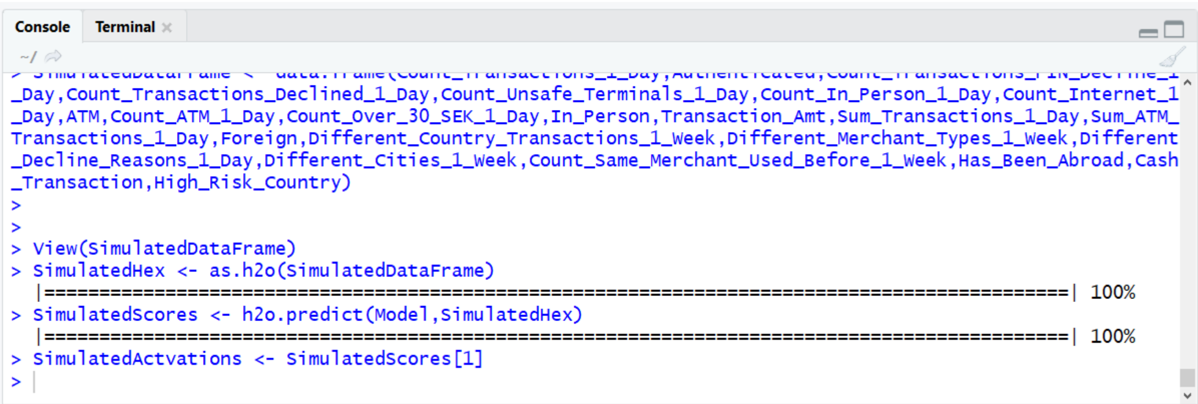


Parse the Activation to a standalone vector:

SimulatedActvations <- as.vector(SimulatedScores[1])

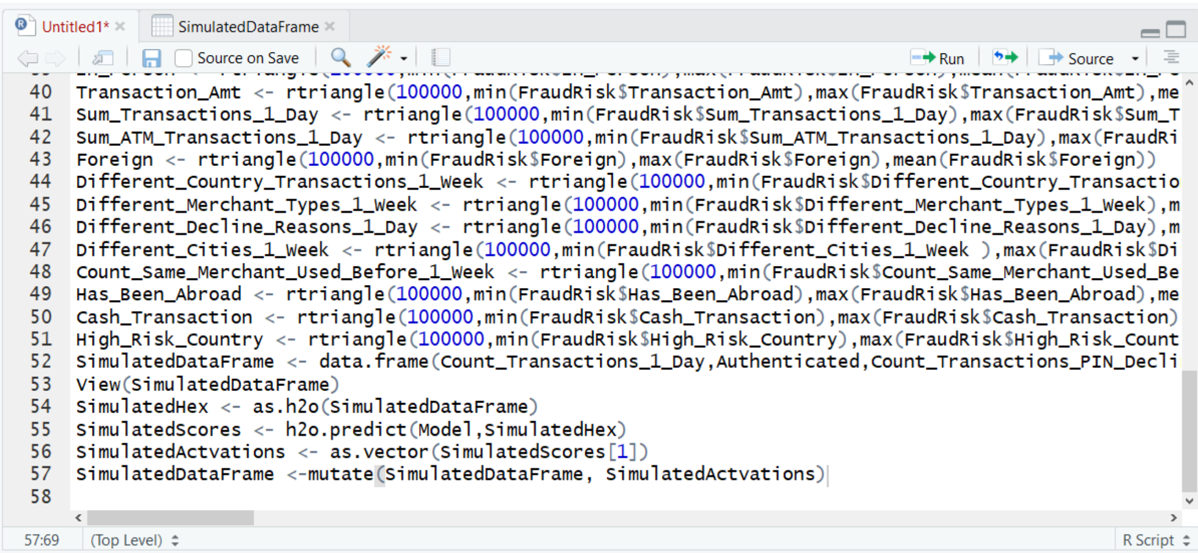


Run the line of script to console:



Append the vector to the simulations data frame (keeping in mind that dplyr is already loaded):

SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)



Run the line of script to console:

```
Console  Terminal
~/
mean(FraudRisk$High_Risk_Country))
> SimulatedDataFrame <- data.frame(Count_Transactions_1_Day,Authenticated,Count_Transactions_PIN_Decline_1_Day,Count_Transactions_Declined_1_Day,Count_Unsafe_Terminals_1_Day,Count_In_Person_1_Day,Count_Internet_1_Day,ATM,Count_ATM_1_Day,Count_Over_30_SEK_1_Day,In_Person,Transaction_Amt,Sum_Transactions_1_Day,Sum_ATM_Transactions_1_Day,Foreign,Different_Country_Transactions_1_Week,Different_Merchant_Types_1_Week,Different_Decline_Reasons_1_Day,Different_Cities_1_Week,Count_Same_Merchant_Used_Before_1_Week,Has_Been_Abroad,Cash_Transaction,High_Risk_Country)
> View(SimulatedDataFrame)
> SimulatedHex <- as.h2o(SimulatedDataFrame)
  |==========================================================================================| 100%
> SimulatedScores <- h2o.predict(Model,SimulatedHex)
  |==========================================================================================| 100%
> SimulatedActvations <- as.vector(SimulatedScores[1])
> SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)
> 
```

Viewing the simulated data frame, scrolling to the last column:

View(SimulatedDataFrame)

Untitled1* | SimulatedDataFrame | Filter

ities_1_Week	Count_Same_Merchant_Used_Before_1_Week	Has_Been_Abroad	Cash_Transaction	High_Risk_Country	SimulatedActvations
2.680810	18.607201	0.52479331	0.2566504	0.84270062	1
2.360935	11.254674	0.33358936	0.8436285	0.53683824	1
2.988296	12.350074	0.50772377	0.6285893	0.75105839	1
3.077607	6.830111	0.43058179	0.8567402	0.38755703	1
4.456409	8.250019	0.44349795	0.4554745	0.72373394	1
2.784764	10.276607	0.77383151	0.4645629	0.27455827	1
3.475589	2.956054	0.23960542	0.6920578	0.21197532	1
2.130021	11.394706	0.41132140	0.6443662	0.11918716	1
1.700184	8.235696	0.66740930	0.3473338	0.10602838	1
3.368588	5.618220	0.62452235	0.4282805	0.16264750	1
3.127950	12.867646	0.49746258	0.7267101	0.39303920	1
3.705824	20.897983	0.12858507	0.3840404	0.38705309	1

Showing 1 to 13 of 100,000 entries

It can be seen that the simulated dataframe has been passed through the H2O neural network as if it were production data. The last column contains the predicted activation, in this case fraud prevention. This data frame can now be used to describe the most likely scenario surrounding an activation.

Procedure 3: Filter Data Frame for Activations and Produce Summary Statistics to prescribe

Keeping in mind that the H2O neural network was trained on real data and is a very good approximation of fraud, by simulating millions of random variables through this model while saving these simulations, it becomes feasible to present summary statistics which can explain what the activation scenario most likely looks like.

The task is to create summary statistics upon the simulations for only those records which have been activated. Start by filtering only those records classified as fraud to a new data frame (keeping in mind dplyr has already been loaded):

SimulatedAndActivated <- filter(SimulatedDataFrame,SimulatedActvations == 1)

```
Sum_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_Transactions_1_Day),max(FraudRisk$Sum_T
Sum_ATM_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_ATM_Transactions_1_Day),max(FraudRi
Foreign <- rtriangle(100000,min(FraudRisk$Foreign),max(FraudRisk$Foreign),mean(FraudRisk$Foreign))
Different_Country_Transactions_1_Week <- rtriangle(100000,min(FraudRisk$Different_Country_Transactio
Different_Merchant_Types_1_Week <- rtriangle(100000,min(FraudRisk$Different_Merchant_Types_1_Week),m
Different_Decline_Reasons_1_Day <- rtriangle(100000,min(FraudRisk$Different_Decline_Reasons_1_Day),m
Different_Cities_1_Week <- rtriangle(100000,min(FraudRisk$Different_Cities_1_Week ),max(FraudRisk$Di
Count_Same_Merchant_Used_Before_1_Week <- rtriangle(100000,min(FraudRisk$Count_Same_Merchant_Used_Be
Has_Been_Abroad <- rtriangle(100000,min(FraudRisk$Has_Been_Abroad),max(FraudRisk$Has_Been_Abroad),me
Cash_Transaction <- rtriangle(100000,min(FraudRisk$Cash_Transaction),max(FraudRisk$Cash_Transaction)
High_Risk_Country <- rtriangle(100000,min(FraudRisk$High_Risk_Country),max(FraudRisk$High_Risk_Count
SimulatedDataFrame <- data.frame(Count_Transactions_1_Day,Authenticated,Count_Transactions_PIN_Decli
View(SimulatedDataFrame)
SimulatedHex <- as.h2o(SimulatedDataFrame)
SimulatedScores <- h2o.predict(Model,SimulatedHex)
SimulatedActvations <- as.vector(SimulatedScores[1])
SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)
View(SimulatedDataFrame)
SimulatedAndActivated <- filter(SimulatedDataFrame,SimulatedActvations == 1)
```

Run the line of script to console:

```
_Day,ATM,Count_ATM_1_Day,Count_Over_30_SEK_1_Day,In_Person,Transaction_Amt,Sum_Transactions_1_Day,Sum_ATM_
Transactions_1_Day,Foreign,Different_Country_Transactions_1_Week,Different_Merchant_Types_1_Week,Different
_Decline_Reasons_1_Day,Different_Cities_1_Week,Count_Same_Merchant_Used_Before_1_Week,Has_Been_Abroad,Cash
_Transaction,High_Risk_Country)
> View(SimulatedDataFrame)
> SimulatedHex <- as.h2o(SimulatedDataFrame)
  |==========================================================================================| 100%
> SimulatedScores <- h2o.predict(Model,SimulatedHex)
  |==========================================================================================| 100%
> SimulatedActvations <- as.vector(SimulatedScores[1])
> SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)
> View(SimulatedDataFrame)
> SimulatedAndActivated <- filter(SimulatedDataFrame,SimulatedActvations == 1)
>
```

The SimulatedAndActivated data frame is now a picture of the activated scenario only, henceforth a series of summary statistics can be executed against this dataframe to begin to understand the environment of fraud. In the following example, a summary of the Count_Transactions_1_Day is provided:

summary(Count_Transactions_1_Day)

```
Sum_ATM_Transactions_1_Day <- rtriangle(100000,min(FraudRisk$Sum_ATM_Transactions_1_Day),max(FraudRi
Foreign <- rtriangle(100000,min(FraudRisk$Foreign),max(FraudRisk$Foreign),mean(FraudRisk$Foreign))
Different_Country_Transactions_1_Week <- rtriangle(100000,min(FraudRisk$Different_Country_Transactio
Different_Merchant_Types_1_Week <- rtriangle(100000,min(FraudRisk$Different_Merchant_Types_1_Week),m
Different_Decline_Reasons_1_Day <- rtriangle(100000,min(FraudRisk$Different_Decline_Reasons_1_Day),m
Different_Cities_1_Week <- rtriangle(100000,min(FraudRisk$Different_Cities_1_Week ),max(FraudRisk$Di
Count_Same_Merchant_Used_Before_1_Week <- rtriangle(100000,min(FraudRisk$Count_Same_Merchant_Used_Be
Has_Been_Abroad <- rtriangle(100000,min(FraudRisk$Has_Been_Abroad),max(FraudRisk$Has_Been_Abroad),me
Cash_Transaction <- rtriangle(100000,min(FraudRisk$Cash_Transaction),max(FraudRisk$Cash_Transaction)
High_Risk_Country <- rtriangle(100000,min(FraudRisk$High_Risk_Country),max(FraudRisk$High_Risk_Count
SimulatedDataFrame <- data.frame(Count_Transactions_1_Day,Authenticated,Count_Transactions_PIN_Decli
View(SimulatedDataFrame)
SimulatedHex <- as.h2o(SimulatedDataFrame)
SimulatedScores <- h2o.predict(Model,SimulatedHex)
SimulatedActvations <- as.vector(SimulatedScores[1])
SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)
View(SimulatedDataFrame)
SimulatedAndActivated <- filter(SimulatedDataFrame,SimulatedActvations == 1)
summary(Count_Transactions_1_Day)
```

Run the line of script to console:

```
_Decline_Reasons_1_Day,Different_Cities_1_Week,Count_Same_Merchant_Used_Before_1_Week,Has_Been_Abroad,Cash
_Transaction,High_Risk_Country)
> View(SimulatedDataFrame)
> SimulatedHex <- as.h2o(SimulatedDataFrame)
  |==========================================================================================| 100%
> SimulatedScores <- h2o.predict(Model,SimulatedHex)
  |==========================================================================================| 100%
> SimulatedActvations <- as.vector(SimulatedScores[1])
> SimulatedDataFrame <-mutate(SimulatedDataFrame, SimulatedActvations)
> View(SimulatedDataFrame)
> SimulatedAndActivated <- filter(SimulatedDataFrame,SimulatedActvations == 1)
> summary(Count_Transactions_1_Day)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.051   6.366   9.955  10.805  14.657  25.980 
> 
```

In this case, it would seem that the average number of transactions on a fraudulent account is 10. When taken in conjunction with other such summary statistics and used in conjunction with the original summary statistics observed from the simulated dataset, this can provide compelling prescriptions.