Getting Up and Running with R in Power BI

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\10_IntroToR\Lab

Lab Overview: This set of lab exercises has been designed to get you up and running with the R data analytic platform and also to get you started with integrating R scripts into the projects you create with Power BI Desktop. You will begin by installing Microsoft R Open to configure your Windows PC with the R platform. Next, you will install a free version of the R development environment named RStudio so that you can become familiar with this popular tool which makes it much easier to write, test and debug scripts written in the R programming language. After that, you will begin to work with R scripts inside a Power BI Desktop project. First, you will learn how to create a query in Power BI Desktop that uses an R scripts as its underlying data source. After that, you will learn how to write R code inside R script visuals which make it possible to generate advanced charts and graphs in a Power BI report.

Exercise 1: Install Microsoft R Open and RStudio

In this exercise you will install the required software to begin working with R. You will start by installing Microsoft R Open which will configure your PC with Microsoft's version of the R analytics platform. After that, you will install the RStudio development environment.

1. Install Microsoft R Open.

   a) Using the browser, navigate to Microsoft Open R Download page located at the following URL.

      https://mran.microsoft.com/download/

   b) Locate and click on the **R Open / MKL** button for the Windows platform to download the MSI installation file.

Note that Microsoft R Open is only available for 64-bit platform. If you are running a 32-bit version of Windows you will not be able to install Microsoft R Open. However, you can install a 32-bit version of R from the Comprehensive R Archive Network (CRAN). Navigate to the page at https://cran.r-project.org/ for instructions to install a 32-bit version of R.

   c) If prompted to save the Microsoft Open R MSI installation file, click **Save**.
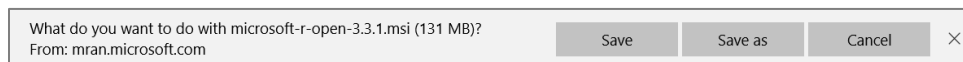
   d) After the Microsoft Open R MSI installation file has downloaded, click the **Run** button to begin the installation.
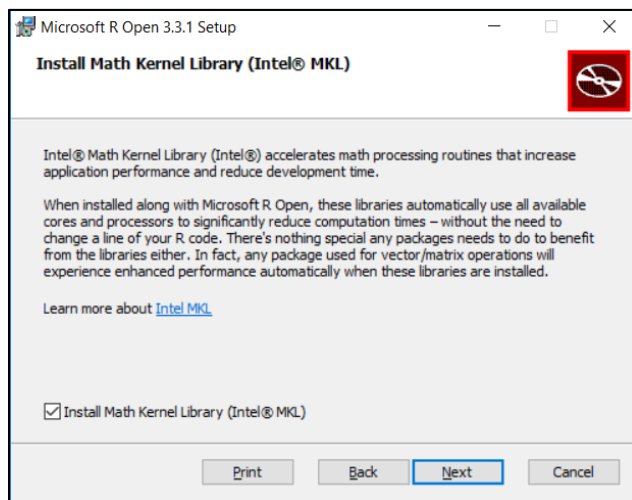

   e) When you see the **Welcome** page of the Microsoft R Open Setup Wizard, click **Next** to continue.

f) On the **Licensing Information** page, check the *I acknowledge* checkbox and then click **Next** to continue.
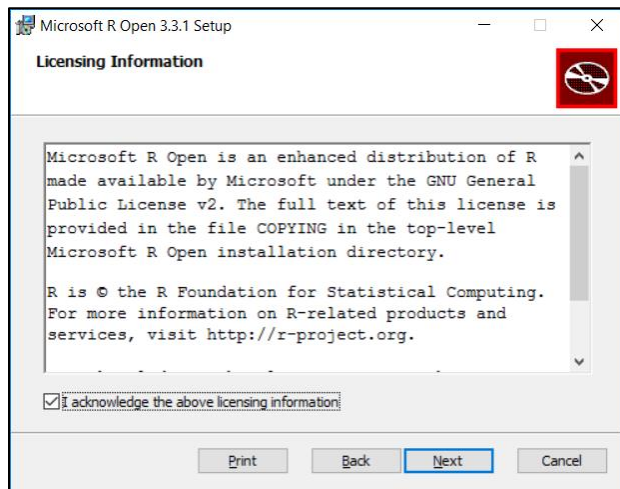
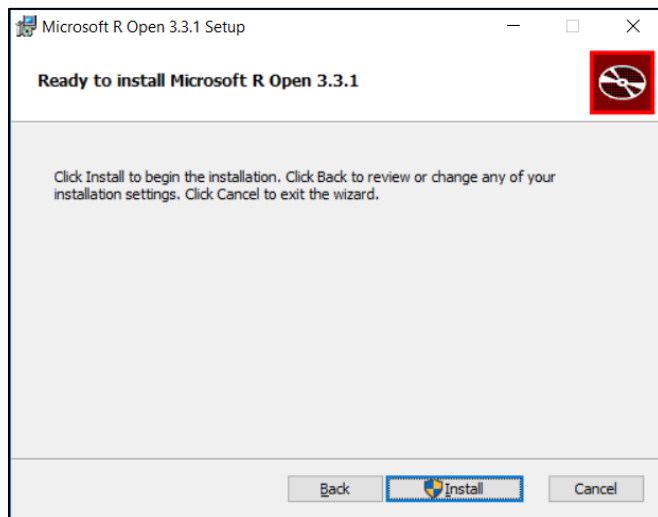g) On the **Install Math Kernel Library** page, leave the checkbox checked and click **Next** to continue.

h) On the **MKL Licensing Information** page, check the *I accept* checkbox and then click **Next** to continue.
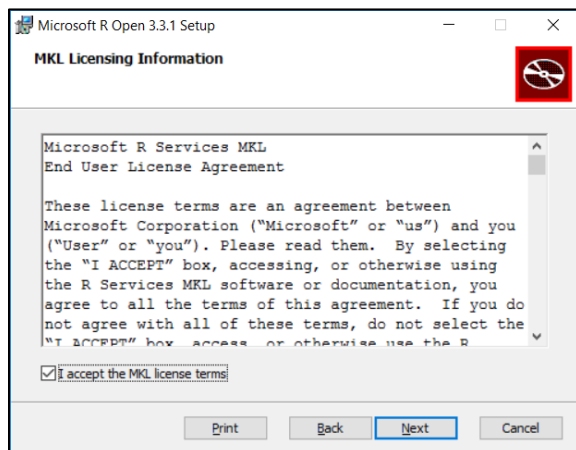
i) On the **Destination Folder** page, accept the defaults and click **Next** to continue.
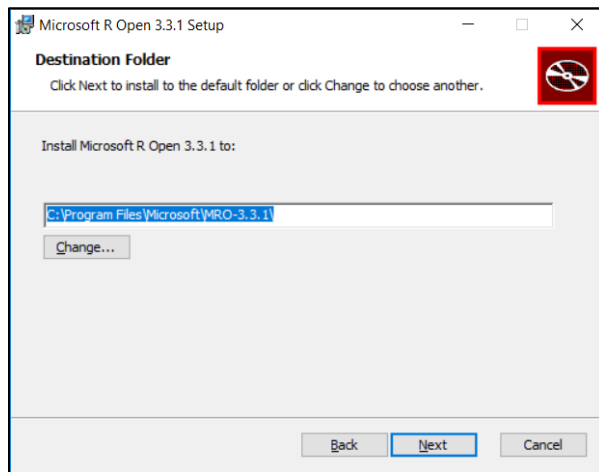
j) On the **Ready to Install Microsoft R Open** page, click **Install** to begin the installation.


k) Wait until you see the **Completed the Microsoft R Open Setup Wizard** dialog.

l) Click the **Finish** button to complete the installation.
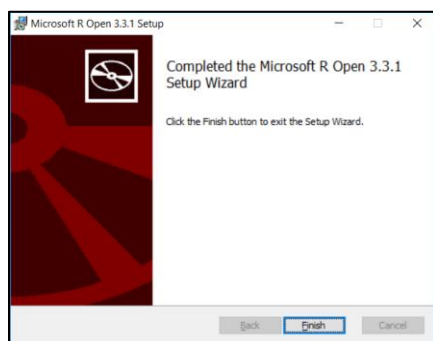

m) If prompted with a dialog to install the **Microsoft Visual C++ 2015 Redistributable**, check the *I agree* checkbox and then click the **Install** button. Wait until the installation is completed and then close the final dialog.
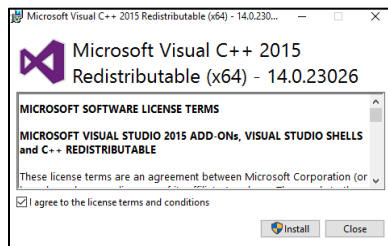
2. Verify the R platform installation by executing R code in the Microsoft R Open Console Application.

   a) Press **Windows** key on the keyboard to display the Windows Start page.

   b) Locate and click the start menu item for **Microsoft R Open** to launch the **RGui** console application.
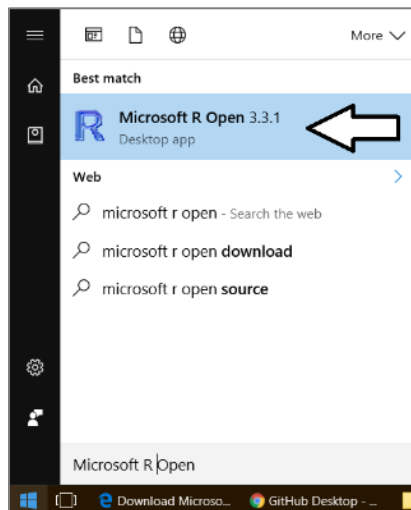
   c) The **RGui** console application should launch and appear as it does in the following screenshot.
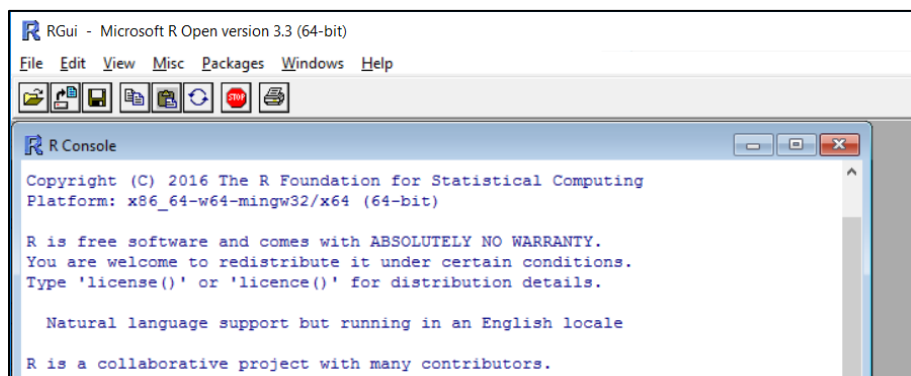
   d) Using the mouse, click in the console windows where the cursor is so you can begin typing.
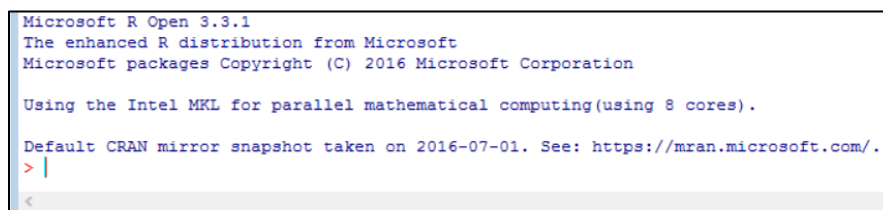
e) Press the **Ctrl + L** keyboard combination to clear the console.

f) Type the following statement into the console and then press **ENTER**.

```
message <- "Hello world"
```
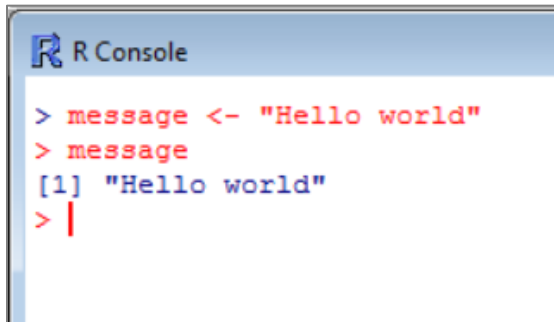
g) Type the following statement into the console and then press **ENTER**.

```
message
```

Remember that the R programming language is case-sensitive. This example uses all lower case letters.

h) You should see that the console returns the value of **[1] "Hello World"** which is a one-element string vector.

i) You have now verified the installation of the R platform. Close the Microsoft R Open console application.

While you could continue to use the R Console application to execute lines of R code, you will find that using RStudio instead of the R Open Console application is much more satisfying.

3. Install RStudio Desktop.

a) Using the browser, navigate to the download page for RStudio Desktop.

```
https://www.rstudio.com/products/rstudio/download3/
```

b) When you get to the download page for RStudio Desktop, scroll down to locate the links for installation files.
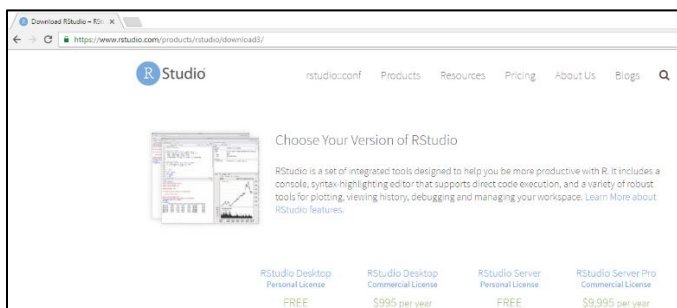
c) Click the link to download the installation files for the **Windows Vista/7/8/10** platform.

d) If prompted to save the RStudio installation EXE file, click **Save**.

What do you want to do with RStudio-0.99.903.exe (77.0 MB)? From: download1.rstudio.org	Save	Save as	Cancel	✕

e) After the RStudio installation EXE file has downloaded, click the **Run** button to begin the installation.

RStudio-0.99.903.exe finished downloading.	Run	Open folder	View downloads	✕

f) When you see the Welcome page of the RStudio Setup Wizard, click **Next** to continue
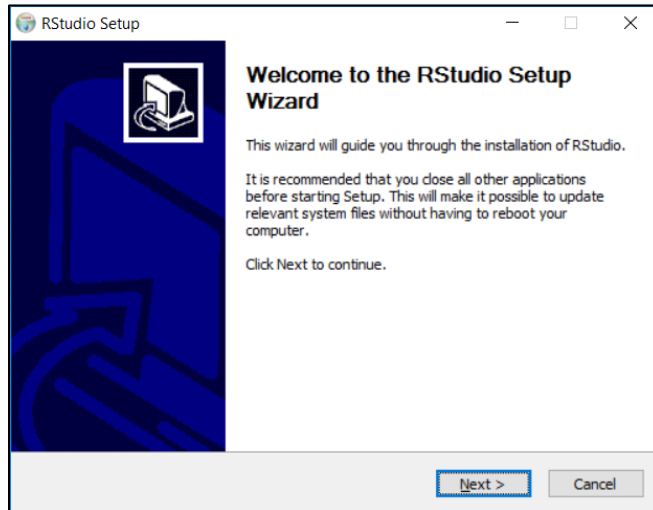
g) On the **Choose Install Location** page, accept the default settings and click **Next** to continue
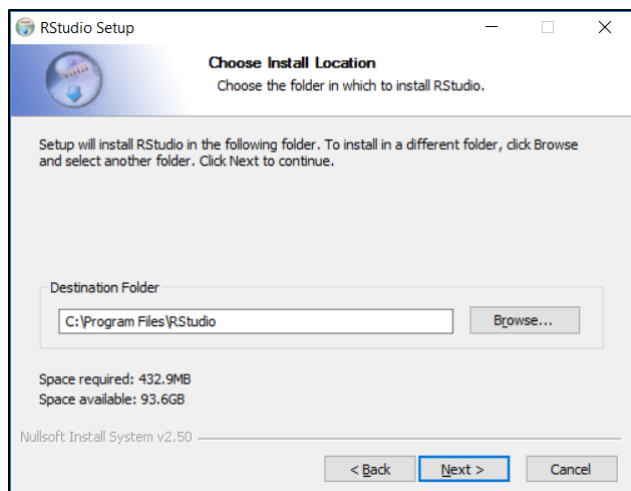
h) On the **Choose Start Menu Folder** page, accept the default settings and click **Install** to begin the installation.
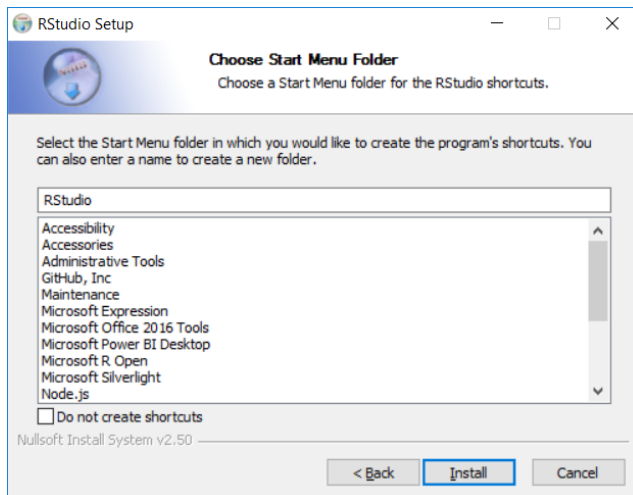
i) Wait until you see the **Completed the RStudio Setup Wizard** dialog.

j) Click the **Finish** button to complete the installation
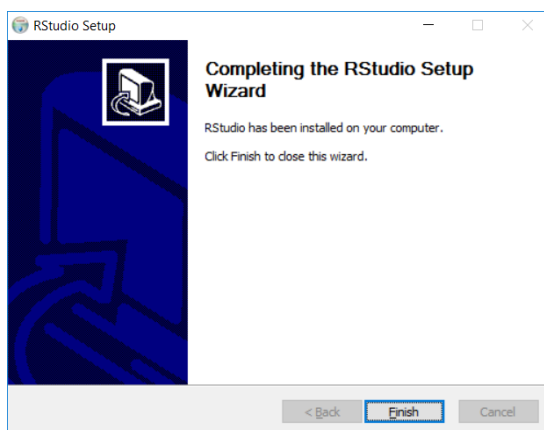
4. Verify the RStudio installation by launching it and executing R code in the RStudio console.

a) Press **Windows** key on the keyboard to display the Windows Start page.

b) Locate and click the start menu item for **RStudio** to launch this application.
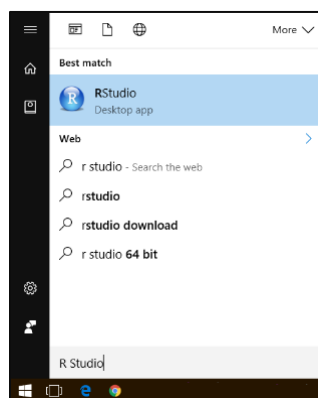
c) When RStudio launches, it should appear with a main application windows which contains several inner sections as shown in the following screenshot.
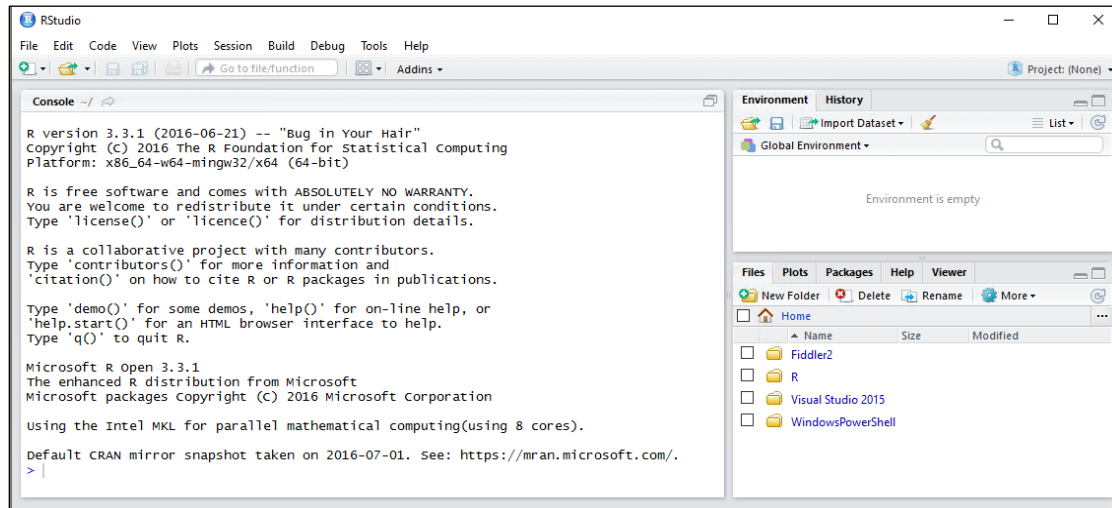
d) Using the mouse, click in the console window in the bottom left where the cursor is so you can begin typing.
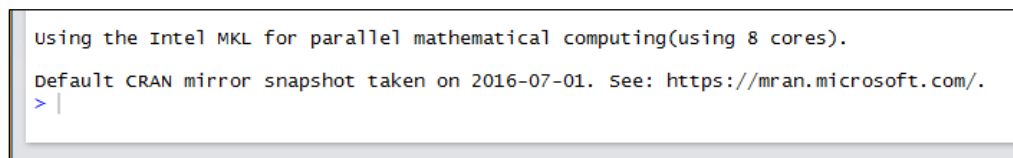
e) Press the **Ctrl + L** keyboard combination to clear the console.

f) Type the following statement into the console and then press **ENTER**.

```
message <- "Hello RStudio"
```

g) Type the following statement into the console and then press **ENTER**.

```
message
```

h) You should see that the console returns the value of **[1] "Hello RStudio"** in the console.

i) In the **Values** section of the **Global Environment** panel, you can see that **message** has been created as a global variable in the current R workspace.
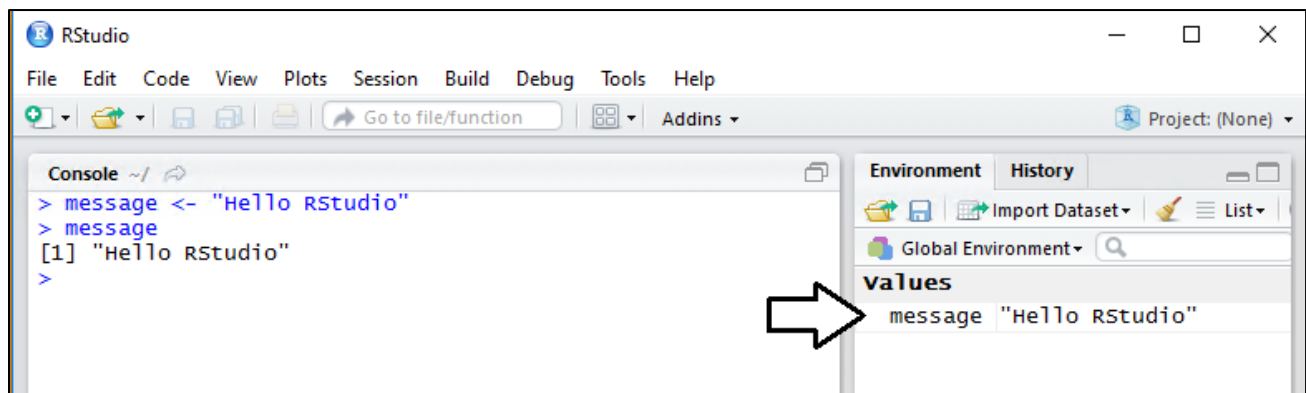
Now that you have created the global object named **message** in your current workspace, you will learn how to delete all your global objects by simply clicking a button inside RStudio.

j) In the **Environment** window which displays global workspace objects, locate and click on the button with the paintbrush icon. This will remove the **message** object and any other objects that have been added to the current workspace.

You have now verified the installation of RStudio. Leave RStudio open as you will use it in the next exercise.
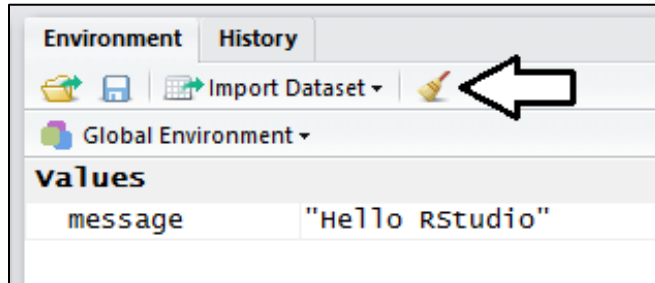
Exercise 2: Create and Execute R Scripts in R-Studio

In this exercise you will begin creating and testing R scripts using the RStudio integrated development environment (IDE). This will give a chance to become familiar with the syntax of the R programming language as well as teach you the fundamental skills for writing and executing R scripts in RStudio. In the final steps of this exercise you will the visualization packages available in R to create a few charts and graphs.

5. Make sure RStudio is still open.

    a) Launch RStudio if it is not already running.

6. Create a new R script.

    a) Select the **File > New File > R Script** menu command or using the **Ctrl+Shift+N** keyboard shortcut to create a new R script.
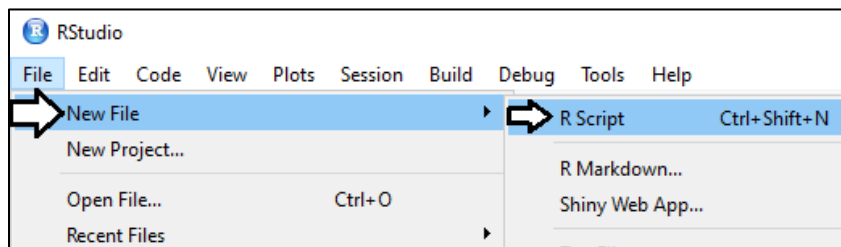
    b) Save the new script as a file named **MyFirstScript** in your Student folder at **C:\Student\Modules\11_IntroToR\Lab**.
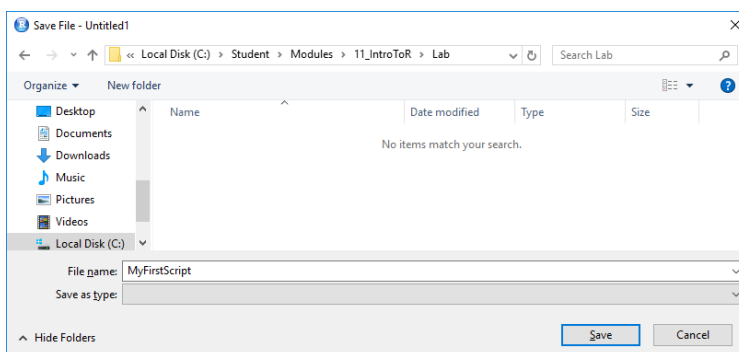
When you initially save a script file that RStudio will add the ***.R** file extension. The script file should be named **MyFirstScript.R**.

7. Execute code in an R script line by line.

   a) Add the following two lines of R code to the top of the script.

   ```
   message <- "Hello R Scripting"
   print(message)
   ```

   b) Place your cursor in the top line of code and then press the **Ctrl+ENTER** keyboard combination to execute that line of code.

   c) You should be able to verify that the line of code executed by examining the **Environment** section and observing that the **message** object that has been created.

When you execute a line of code using the **Ctrl+Enter** keyboard shortcut, RStudio automatically advances the cursor to the next line of code. That means you can repeatedly press the **Ctrl+Enter** keyboard shortcut to execute line after line in a script.

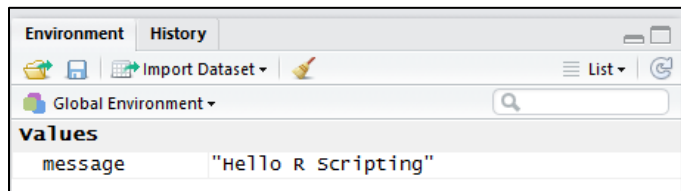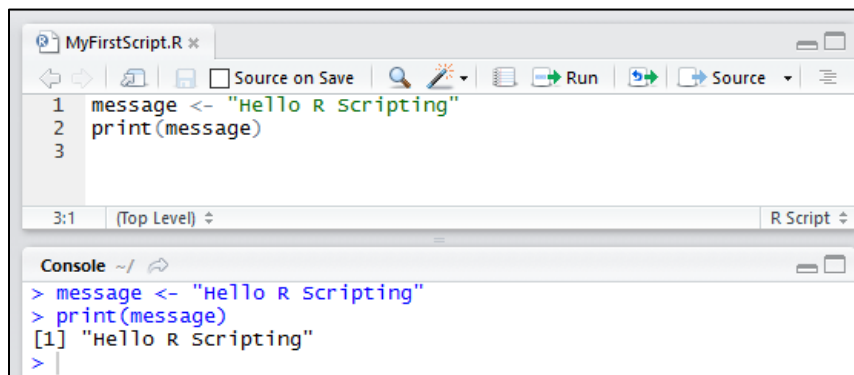   d) Press the **Ctrl+Enter** keyboard shortcut a second time to execute the line of code that calls the **print** function.

   e) Examine the console window to see the results of the **print** function.

When you execute R code inside a script in a line-by-line fashion using the **Ctrl+ENTER** keyboard combination, you can see that each line of code that is executed is echoed as output in the console window.

8. Your cursor should now be located in the script window. In this step you will learn how to move back and forth between the script window and the console window using keyboard shortcuts.

   a) Press the **Ctrl+2** keyboard combination to move to the console window.

   b) Press the **Ctrl+1** keyboard combination to move back to the script window.

As you can see, it's very easy to move between these two windows without having to use the mouse.

9. Clear the Console window.

   a) Press the **Ctrl+2** keyboard combination to move to the console window.

   b) Press the **Ctrl + L** keyboard combination to clear the console window.

   c) Press the **Ctrl+1** keyboard combination to move back to the script window.

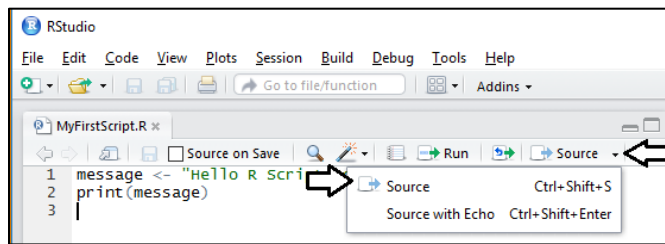10. Execute the code in an R script as a single batch.

   a) Drop down the **Source** menu button and select the **Source** command. Alternatively, you can execute the R script in the active window by using the **Ctrl+Shift+S** keyboard combination.

b) After the script has executed, examine the console window. You will see that RStudio executed the **source** function to run the entire script as a batch.
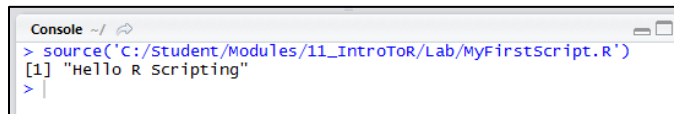
11. Save and close the script named **MyFirstScript.R**.

12. Write R code to create and manipulate a vector.

a) Use the **Ctrl+Shift+N** keyboard shortcut to create a new R script.

b) Save the new script as a file named **FunWithVectors** in your Student folder at **C:\Student\Modules\11_IntroToR\Lab**.

c) Add the following code to create and print a vector which contains a set of whole numbers for recorded average temperatures.

```
averageTempature <- c(74, 80, 64, 52, 36, 28)
print(averageTempature)
```

d) Execute the script and examine the output in the console window.
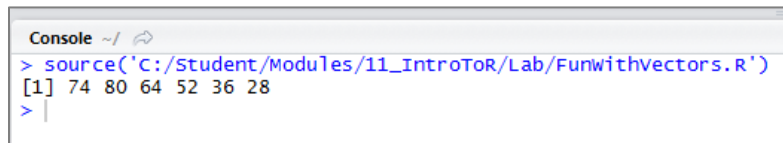
e) Add the following two new lines below the first two lines to add names to each vector element and print out the vector again.

```
averageTempature <- c(74, 80, 64, 52, 36, 28)
print(averageTempature)

names(averageTempature) = c("JUL", "AUG", "SEP", "OCT", "NOV", "DEC")
print(averageTempature)
```

f) Execute the script and examine the output in the console window. You should observe that the vector elements have been printed out with their names.
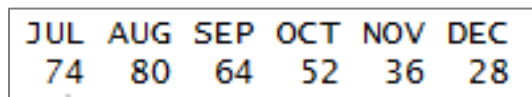
g) Add the following code to the bottom of the script to create a new vector named **averageTempatureQ3** which subsets the **averageTempature** to include the first 3 months.

```
averageTempatureQ3 = averageTempature[c(1, 2, 3)]
print(averageTempatureQ3)
```

h) Add the following code to the bottom of the script to create a new vector named **averageTempatureQ4** which subsets the **averageTempature** to exclude the first 3 months.

```
averageTempatureQ4 = averageTempature[-c(1, 2, 3)]
```

```
print(averageTempatureQ4)
```

i) Add the following code to the bottom of the script to create a new vector named **coldMonths** which subsets the **averageTempature** to include all months with an average temperature below 50 degrees.

```
coldMonths = averageTempature[averageTempature<50]
print(coldMonths)
```

j) Your script should now contain code which matches the following code listing.

```
averageTempature <- c(74, 80, 64, 52, 36, 28)
print(averageTempature)

names(averageTempature) = c("JUL", "AUG", "SEP", "OCT", "NOV", "DEC")
print(averageTempature)

averageTempatureQ3 = averageTempature[c(1, 2, 3)]
print(averageTempatureQ3)

averageTempatureQ4 = averageTempature[-c(1, 2, 3)]
print(averageTempatureQ4)

coldMonths = averageTempature[averageTempature<50]
print(coldMonths)
```

k) Execute the script and examine the output in the console window. You should see in the console that matches the following screenshot.

```
JUL  AUG  SEP
 74   80   64
OCT  NOV  DEC
 52   36   28
NOV  DEC
 36   28
```

In the next step you will use vector arithmetic to convert the temperature values from Fahrenheit to Celsius. You can convert a temperature value from Fahrenheit to Celsius by subtracting 32 and then multiplying the difference by 5/9.

l) Add the following code to the bottom of the script to create a new vector with temperatures converted to Celsius and to assign the new vector to a new object named **averageTempatureCelsius**.

```
averageTempatureCelsius <- (averageTempature-32) * (5/9)
print(averageTempatureCelsius)
```

m) Execute the script and examine the output in the console window. You should observe the elements in the new vector have been converted to Celsius. However, they are also showing 6 points of precision after the decimal point which is distracting.

```
      JUL       AUG       SEP       OCT       NOV       DEC
23.333333 26.666667 17.777778 11.111111  2.222222 -2.222222
```

n) Modify your code by adding the **round** function to round each temperature to w whole number.

```
averageTempatureCelsius <- round( (averageTempature-32) * (5/9), 0)
print(averageTempatureCelsius)
```

o) Execute the script and examine the output in the console window. You should observe that the elements in the new vector display Celsius values that have been rounded to whole numbers.

JUL	AUG	SEP	OCT	NOV	DEC
23	27	18	11	2	-2

p) Save and close the script file **FunWithVectors.R**.

13. Clear the console window and remove all existing objects from the global environment.

   a) Make sure your cursor is in the console window.

   b) Press the **Ctrl + L** keyboard combination to clear the console window.

   c) Click the button with the paintbrush icon in the **Environment** window to remove global objects from the current workspace.
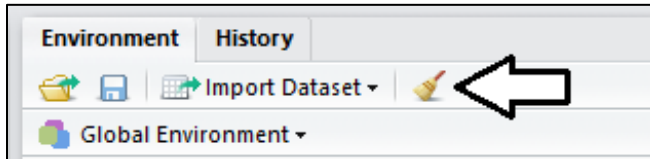

In the next step you will begin to work with data frames by importing data from a CSV file named **SalesByCustomer.csv** that is located in the GitHub repository for Power BI Bootcamp. The file named **SalesByCustomer.csv** can be downloaded using a standard HTTP GET request using anonymous access at the following URL:

https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesByCustomer.csv

14. Load a data frame of customer data from a CSV file that is accessible over the Internet.

   a) Use the **Ctrl+Shift+N** keyboard shortcut to create a new R script.

   b) Save the new script as a file named **CustomerAnalysis** in your Student folder at **C:\Student\Modules\11_IntroToR\Lab**.

   c) Add the following code to download a CSV file and convert its data into a data frame.

```
urlData = "https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesByCustomer.csv"
customers <- read.csv(urlData)
```

   d) Execute the script and examine the output in the console window.

```
Console ~/ 
> source('C:/Student/Modules/11_IntroToR/Lab/CustomerAnalysis.R')
> |
```
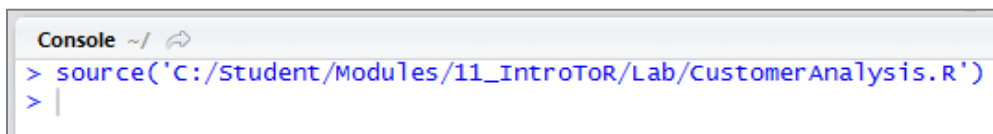
The console window should just show a single line which indicates the **source** function was used to execute the script. The main thing to verify in this case is that the script ran and imported the data from the CSV file without any errors. If there were any errors during script execution, you would see error messages in the console window. If you don't see any errors, you can assume that the data import operation to create a new data frame was successful.

   e) Examine the **Environment** window. You should be able to verify that a new object has been created in the **Data** section named **customers**. This object is a data frame that has been created from the data that has been imported from the CSV file.
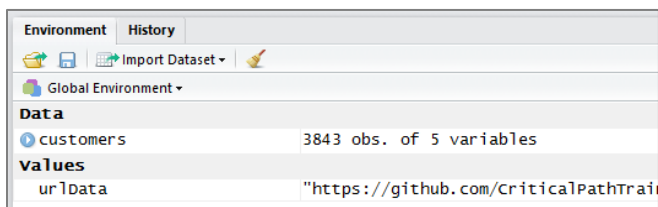
   f) Using our mouse, double-click on **customers** in the **Global Environment** section to view it inside the data frame viewer.

g) Inspect the **customers** data frame in the viewer provided by RStudio. What you see should match the following screenshot.

h) Add the following line of code to the bottom of your script to assign user-friendly names to each column in the data frame.

```
urlData = "https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesByCustomer.csv"
customers <- read.csv(urlData)

names(customers) = c("Customer","State","Gender","Age","Sales")
```

i) After you have added the code to modify the column names, rerun the script and then examine the **customers** data frame in the viewer to verify that the columns names have been updated.

j) Add the following line of code to the bottom of your script to sort the data frame by the **Sales** column in a descending fashion.

```
urlData = "https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesByCustomer.csv"
customers <- read.csv(urlData)

names(customers) = c("Customer","State","Gender","Age","Sales")

customers <- customers[ order(customers$Sales, decreasing=TRUE),  ]
```
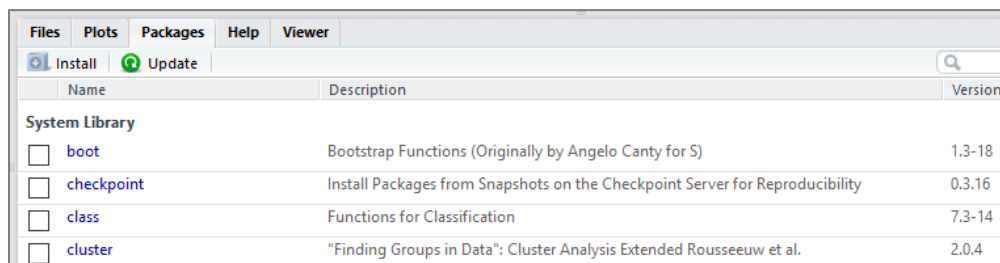
k) After you have added the code to sort the data frame, rerun the script and then examine the **customers** data frame in the viewer to verify that the rows have been sorted by the **Sales** column in a descending fashion.

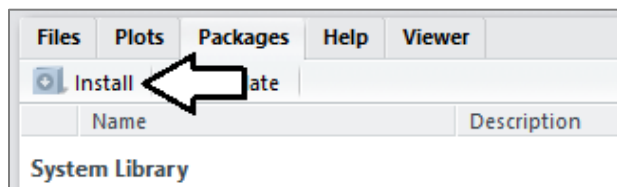|	Customer	State	Gender	Age	Sales
620	Cedrick Beck	FL	M	81	2080
2148	Kristopher Blankenship	FL	M	47	1789
3621	Tonia Hughes	FL	F	75	1638
64	Alberta Odonnell	GA	F	42	1630
2584	Mariana William	OH	F	58	1549
1693	Jacinto Fuentes	NY	M	30	1540
1664	Irwin Sampson	FL	M	43	1520
115	Alonzo Hoffman	TX	M	77	1519

Over the next few steps, you will add the code to convert values from the **Gender** column to make them more readable. More specifically, you will convert values of "F" to "Female" and values of "M" to "Male". However, you will use a convenient R function named **mapvalues** that is in a R package named **plyr** that is not installed by default. Therefore, you must ensure the **plyr** package is installed before you can successfully execute your code.

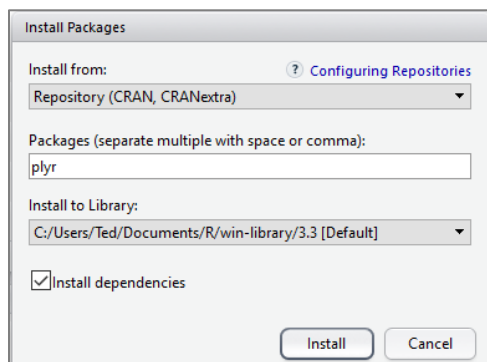15. Install the **plyr** package by hand.

   a) Select the **Packages** tab in the bottom right corner window to see the list of installed packages.

Files	Plots	Packages	Help	Viewer		
● Install	● Update					Q
	Name		Description			Version
System Library						
☐	boot		Bootstrap Functions (Originally by Angelo Canty for S)			1.3-18
☐	checkpoint		Install Packages from Snapshots on the Checkpoint Server for Reproducibility			0.3.16
☐	class		Functions for Classification			7.3-14
☐	cluster		"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.			2.0.4

   b) Click on the **Install** button in the **Packages** window.

Files	Plots	Packages	Help	Viewer	
● Install ←	ate				
	Name			Description	
System Library					

   c) In the **Packages** textbox, type **plyr** and then click the Install button.

Install Packages

Install from: ? Configuring Repositories
Repository (CRAN, CRANextra) ▼

Packages (separate multiple with space or comma):
plyr

Install to Library:
C:/Users/Ted/Documents/R/win-library/3.3 [Default] ▼

☑ Install dependencies

[Install] [Cancel]

   d) When you click the **Install** button, RStudio will echo the code it is running in the console. By examining the console output, you should see that installing the **plyr** package also results in the installation of a dependent package named **Rcpp**.

```
> install.packages("plyr")
Installing package into 'C:/Users/Ted/Documents/R/win-library/3.3'
(as 'lib' is unspecified)
also installing the dependency 'Rcpp'

trying URL 'https://mran.revolutionanalytics.com/snapshot/2016-07-01/bin/windows/contrib/3.3/Rcpp_0.12.5.zip'
content type 'application/zip' length 3225427 bytes (3.1 MB)
downloaded 3.1 MB

trying URL 'https://mran.revolutionanalytics.com/snapshot/2016-07-01/bin/windows/contrib/3.3/plyr_1.8.4.zip'
content type 'application/zip' length 1182167 bytes (1.1 MB)
downloaded 1.1 MB

package 'Rcpp' successfully unpacked and MD5 sums checked
package 'plyr' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Ted\AppData\Local\Temp\RtmpiURFrr\downloaded_packages
> |
```

e) After the package installation is complete, you should see the two packages **plyr** and **Rcpp** displayed in the **User Library** section of the **Packages** window.

Files	Plots	Packages	Help	Viewer	
Install	Update				
	Name		Description		
User Library					
☐	plyr		Tools for Splitting, Applying and Combining Data		
☐	Rcpp		Seamless R and C++ Integration		
System Library					
☐	boot		Bootstrap Functions (Originally by Angelo Canty for S)		
☐	checkpoint		Install Packages from Snapshots on the Checkpoint Server for Reproducibility		
☐	class		Functions for Classification		
☐	cluster		"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.		

f) Note that you can add code to an R script to automatically install a package using the **install.packages** function. The most common approach is to write code that determines whether the package is already installed and then automatically installs the package if it is not already installed as shown in the following R code listing.

```
# install plyr package if it is not currently installed
if(!('plyr' %in% rownames(installed.packages()))){
   install.packages("plyr")
}
```

Now that the **plyr** package has been installed, you can now load this package into the current workspace using the **library** function. Once the package has been loaded, you can begin to use the **mapvalues** function.

16. Add code to remap **Gender** column values from "F" to "Female" and from "M" to "Male".

   a) At the bottom of the script, add a call to **library(plyr)** to load the **plyr** package into the current workspace.

   b) Add a call to **mapvalues** to remap **Gender** column values from "F" to "Female" and from "M" to "Male"

```
library(plyr)
customers$Gender <- mapvalues(customers$Gender, from = c("F", "M"), to = c("Female", "Male"))
```

   c) After you have added the code to remap the **Gender** column values, rerun the entire **CustomerAnalysis** script and then examine the **customers** data frame in the viewer. You should be able to verify that the **Gender** column now contains values of "Female" and "Male" instead of "F" and "M".

	Customer	State	Gender	Age	Sales
620	Cedrick Beck	FL	Male	81	2080
2148	Kristopher Blankenship	FL	Male	47	1789
3621	Tonia Hughes	FL	Female	75	1638
64	Alberta Odonnell	GA	Female	42	1630
2584	Mariana William	OH	Female	58	1549
1693	Jacinto Fuentes	NY	Male	30	1540
1664	Irwin Sampson	FL	Male	43	1520
115	Alonzo Hoffman	TX	Male	77	1519

In the next step you will write the R code to add a new column to the data frame named **AgeGroup**. Your code will be required to calculate an **AgeGroup** value for each customer by taking all possible **Age** values and breaking them down into 6 discrete groups. This will demonstrate how to use the R function named **cut** to convert a continuous variable such as **Age** into a categorical variable.

17. Add code to add the **AgeGroup** variable to the **customers** data frame.

a) At the bottom of the script, create a new vector named **ageGroups** which contains the integer values of **0,17,29,39,49,64,100**. Note that there are 7 integer values in this vector to define the starting and ending point for 6 discrete age groups.

```
ageGroups = c(0,17,29,39,49,64,100)
```

b) Add the following code to create another vector named **ageGroupLabels** which contains the labels for each of the 6 groups.

```
ageGroupLabels = c("Under 18", "18 to 29", "30 to 39", "40 to 49", "50 to 64", "65 and up")
```

c) Add the following code to use the **cut** function to break the **Age** variable into a set of 6 discrete values. Assign the result of the **cut** function to a new variable in the **customer** data frame named **AgeGroup**.

```
customers$AgeGroup <- cut(customers$Age, breaks = ageGroups, labels = ageGroupLabels)
```

d) Your code should now match the following code listing.

```
ageGroups = c(0,17,29,39,49,64,100)
ageGroupLabels = c("Under 18", "18 to 29", "30 to 39", "40 to 49", "50 to 64", "65 and up")
customers$AgeGroup <- cut(customers$Age, breaks = ageGroups, labels = ageGroupLabels)
```

e) Rerun the entire **CustomerAnalysis** script and then examine the **customers** data frame in the viewer. You should be able to verify that the **AgeGroup** variable now contains values that match the following screenshot.

18. At this point, the code you have written in the R script named **CustomerAnalysis.R** should match the following code listing.

```
urlData = "https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesByCustomer.csv"
customers <- read.csv(urlData)

names(customers) = c("Customer","State","Gender","Age","Sales")

customers <- customers[ order(customers$Sales, decreasing=TRUE),  ]

if(!('plyr' %in% rownames(installed.packages()))){
  install.packages("plyr")
}

library(plyr)
customers$Gender <- mapvalues(customers$Gender, from = c("F", "M"), to = c("Female", "Male"))

ageGroups = c(0,17,29,39,49,64,100)
ageGroupLabels = c("Under 18", "18 to 29", "30 to 39", "40 to 49", "50 to 64", "65 and up")
customers$AgeGroup <- cut(customers$Age, breaks = ageGroups, labels = ageGroupLabels)
```

19. Add code to create a histogram in the default output device.

a) Place the cursor underneath the existing code in **CustomerAnalysis.R** so you can add more code at the end of the script.

b) Create a new data frame named **customers.florida** which subsets the **customers** data frame where **State** equals **FL**.

```
customers.florida = subset(customers, State == "FL", c("Customer", "Age") )
```

c) Add the following code which creates a histogram to visualize customer count by age.

```
hist(customers.florida$Age,
     main = "Florida Customer Count by Age",
     col="lightblue",
     border="black",
     ylab="Customer Count",
     xlab="Customer Age",
     xlim = c(20, 100),
     breaks=10 )
```

d) Rerun the entire **CustomerAnalysis** script. When the code executes to generate the histogram, RStudio will display this visualization in the **Plots** window in the lower right-hand corner of the main RStudio window.

20. Add code to create a barplot in the default output device.

a) Place the cursor under all the existing code in **CustomerAnalysis.R** so you can write additional code at the end of this script.

b) Create a new data frame named **customerSales.florida** which subsets the **customers** data frame with the **AgeGroup** variable and **Sales** variable and filters customers for the state of Florida (FL).

```
customerSales.florida <- subset(customers, State == "FL", c( "AgeGroup", "Sales"))
```

c) Use the **aggregate** function to sum the **Sales** variable values by the **AgeGroup** variable.

```
customerSales.florida <-
  aggregate(customerSales.florida$Sales , by=list(AgeGroup=customerSales.florida$AgeGroup), sum)
```

d) Add the following code to generate a barplot.

```
barplot(salesByAgeGroup$x,
        names.arg = salesByAgeGroup$AgeGroup,
        ylab = "Sales Revenue",
        main = "Florida Sales by Customer age",
        col = c("red", "yellow","orange",  "green", "blue") )
```

e) The code at the bottom of **CustomerAnalysis.R** should now match the following code listing.

```
customerSales.florida <- subset(customers, State == "FL", c( "AgeGroup", "Sales"))

customerSales.florida <-
  aggregate(customerSales.florida$Sales , by=list(AgeGroup=customerSales.florida$AgeGroup), sum)

barplot(salesByAgeGroup$x,
        names.arg = salesByAgeGroup$AgeGroup,
        ylab = "Sales Revenue",
        main = "Florida Sales by Customer Age",
        col = c("red", "yellow","orange",  "green", "blue") )
```

f) Rerun the entire **CustomerAnalysis** script. When the code executes the last line to generate the barplot, RStudio will display it in the **Plots** windows in the lower right-hand side of the main RStudio window.
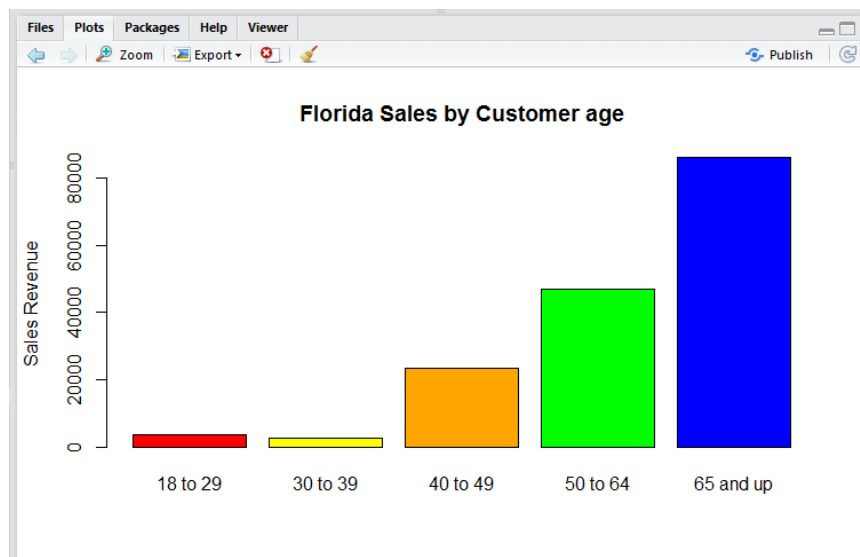
21. Save your changes to **CustomerAnalysis.R**.

Now that you have gotten some hands-on experience writing and testing R code in RStudio, it's time to move forward and begin working with R scripts inside Power BI Desktop.

Exercise 3: Create a Power BI Query using an R Script Data Source

In this exercise you will create a new dashboard using the dataset and report you created in early labs using Power BI Desktop.

1. Launch Power BI Desktop to start a new project.
2. Save the new project as **R Scripting Lab.pbix** using the following path.

```
C:\Student\Projects\R Scripting Lab.pbix
```

3. Create a new query based on an R script.
   a) Drop down the **Get Data** menu button on the ribbon and click **More…**.
   b) In the **Get Data** dialog, select the **Other** option on the left and then select **R Script** and click **Connect**.
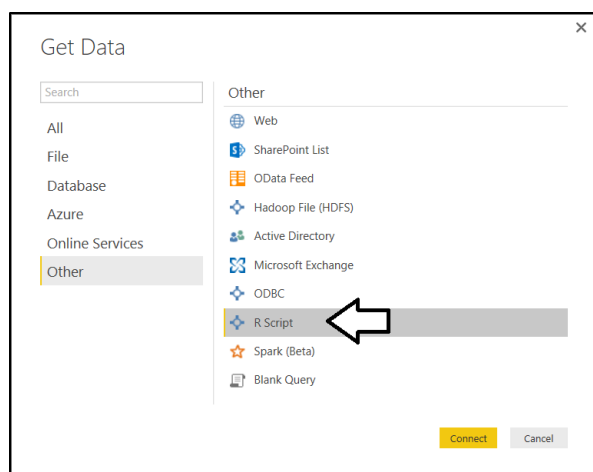
c) At this point, you should see the **Execute R Script** dialog with an empty textbox in which you can type or paste in an R script.

d) Copy the following R code into the Windows clipboard.

```
# create temp file
temp <- tempfile()

# download zip archive to temp file
urlZipArchive = "https://github.com/CriticalPathTraining/PBI365/raw/master/Student/Data/SalesData.zip"
download.file(urlZipArchive ,temp)

# extract CSV file out of zip
csvFile = unz(temp, "SalesByCustomer.csv")

# load new data frame from data in CSV file
customers <- read.csv(csvFile)

# perform cleanup
unlink(temp)
remove(csvFile)
remove(temp)
remove(urlZipArchive)
```

e) Paste the R code from the Widows clipboard into the **Execute R Script** dialog and click **OK**.

f) In the **Navigator** dialog, select the **customers** data frame on the right.

g) At the bottom of the **Navigator** dialog, click the **Edit** button to open the new query in the Query Editor window.
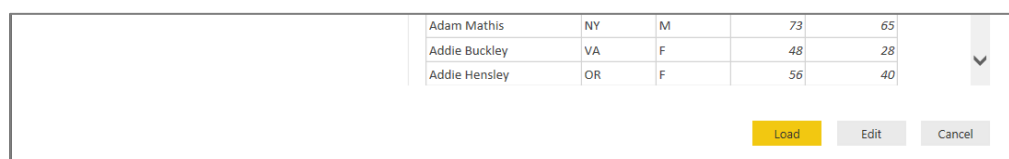
h) At this point, the Query Editor window should display the data that has been imported by the R script.

i) Using the Query Editor window, update the column names to **Customer**, **State**, **Gender**, **Age** and **Sales**.

j) Click the **Close and Apply** button in the ribbon to close the Query Editor window and return to the main application window of Power BI Desktop.

k) Navigate to Data View and inspect the query output in data view. Keep in mind that this data was extracted from the zip archive as the R script executed.

4. Click the **Refresh** button on the **Home** tab of the ribbon. Each time you click the **Refresh** button, Power BI Desktop runs through the entire process of downloading the zip archive to a temporary file, extracting the CSV file from inside and using that CSV file to load a R data frame that is recognized as a table in Power BI Desktop.

5. Modify the query to add a new applied step which executes an R script to generate the **AgeGroup** column.

a) Click on the **Edit Queries** button in the ribbon to reopen the **Query Editor** window.
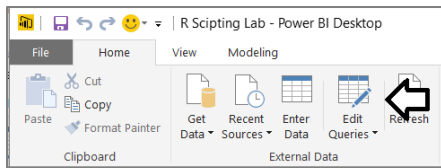
b) Navigate to the **Transform** tab and click the **Run R Script** button on the far right side of the ribbon.
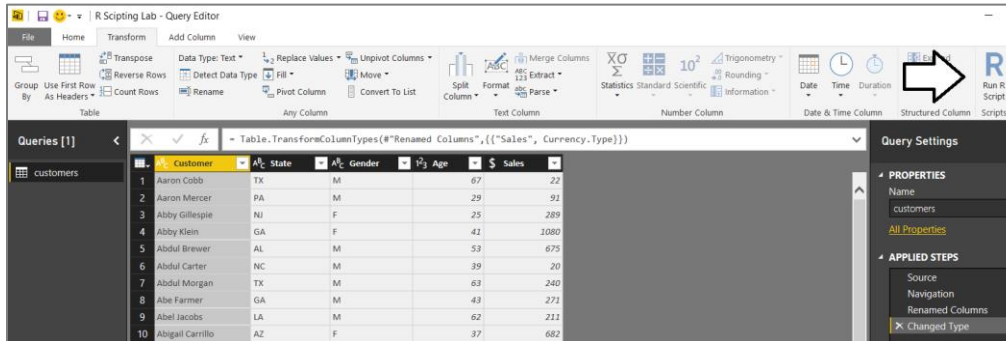
c) The **Run R Script** dialog should appear as shown in the following screenshot.
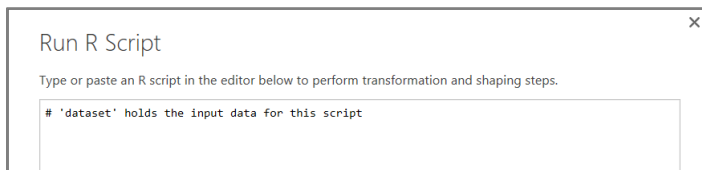
d) Copy and paste the following code into the **Run R Script** dialog.

```
ageGroups = c(0,17,29,39,49,64,100)
ageGroupLabels = c("Under 18", "18 to 29", "30 to 39", "40 to 49", "50 to 64", "65 and up")

dataset$AgeGroup <- cut(dataset$Age, breaks = ageGroups, labels = ageGroupLabels)

output <- dataset
```

e) When the **Run R Script** dialog matches the following screenshot, click **OK** to enter your changes.
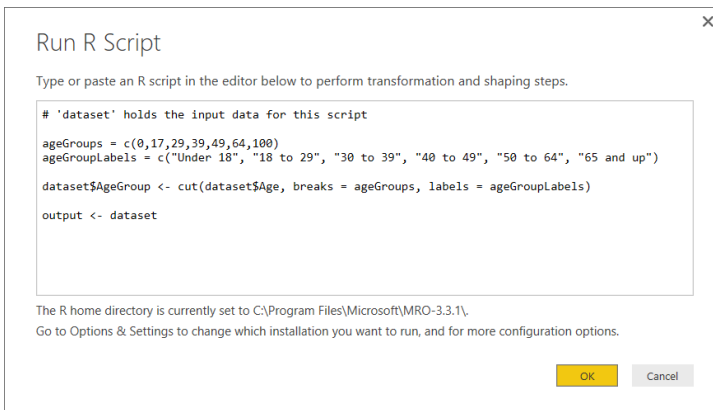
f) You should see that the R script execution has added a new column named **AgeGroup**.

g) Convert the data type of the **Sales** column to **Fixed Decimal Number**.

> R script execution has problems when encountering variables that have been converted to the **Fixed Decimal Number** data type. Therefore, you should execute R scripts first and then convert columns to the **Fixed Decimal Number** data type afterwards.

h) Click the **Close and Apply** button to close the Query Editor window and return to the main application window.

i) Inspect the customers table in Data View to see the new **AgeGroup** column.

6. Save your work by clicking the Save button in the upper right-hand corner of the main window of Power BI Desktop.

Exercise 4: Write R Code to Generate Charts using the R Visual

In this exercise you will design a report using R script visuals. This will give you an opportunity to write R code to create charts and graphs using the R data analytics platform.

1. Make sure the **R Scripting Lab.pbix** project that you created in the previous exercise is still open in Power BI Desktop.

2. Navigate to Report View. There should be a single page in the report named **Page1**.

3. Change the name of the page to **Customer Analysis**.
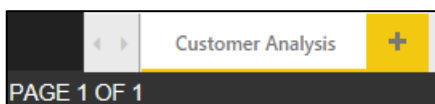
4.   Add a R script visual to the report to create a histogram.

a)   Click the **R** button on the **Visualizations** list to add an R script visual to the report.
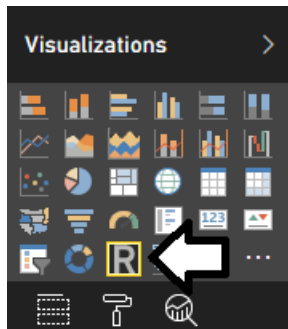
b)   You can see that there is an R script editor that appears beneath the report page when the R script visual is selected.
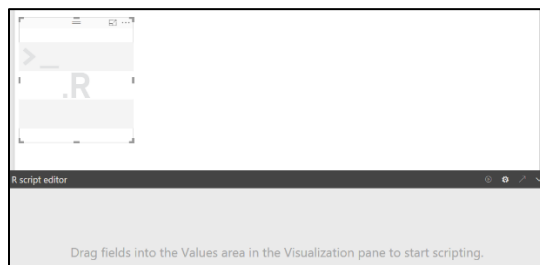
c)   With the R script visual selected, add the **Customer** field and the **Age** field into the **Values** well.
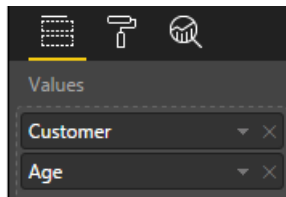
d)   Examine the R script editor. You can see that Power BI desktop has automatically added code to create a data frame named **dataset** which contains two variables based on the **Customer** field and the **Age** field.
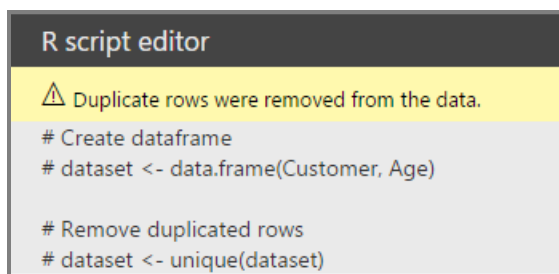
e)   Add the following code to the R script editor window to create a histogram.

```
hist(dataset$Age,
     main = "Customer Count by Age",
     col="lightblue",
     border="black",
     ylab="Customer Count",
     xlab="Customer Age",
     xlim = c(18, 100),
     breaks=20 )
```

f) Once you have added the code, the R script editor window should match the following screenshot.

```
R script editor

⚠ Duplicate rows were removed from the data.
# Create dataframe
# dataset <- data.frame(Customer, Age)

# Remove duplicated rows
# dataset <- unique(dataset)
hist(dataset$Age,
    main = "Customer Count by Age",
    col="lightblue",
    border="black",
    ylab="Customer Count",
    xlab="Customer Age",
    xlim = c(18, 100),
    breaks=20 )

◀ ▶    Customer Analysis    ✚
```

g) Locate the toolbar button in the top right corner of the R script editor and click the **Run** button to execute your R code.

h) When you click the **Run** button, you should see a histogram appear like the one shown in the following screenshot.


i) If there is an error in your R code, you will see the following error message. You can click on the **See details** link to view the error message which will help you to debug your code.

5. Add a second R script visual to generate a barplot.

    a) Add a second R script visual to the page.

    b) With the new R script visual selected, add the **AgeGroup** field and the **Sales** field into the **Values** well.
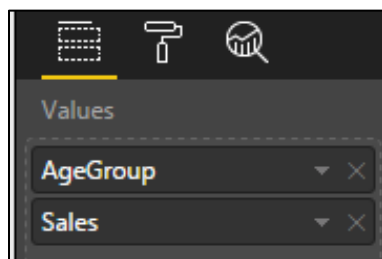

    c) Add the following code to the R script editor window to create a barplot.

```
barplot(dataset$Sales,
        names.arg = dataset$AgeGroup,
        ylab = "Sales Revenue",
        main = "Sales by Customer Age Group",
        col = c("red", "yellow","orange",  "green", "blue"))
```

    d) The R script editor window should match the following screenshot. Click the **Run** button to test your code.

e) When you click the **Run** button, you should see a barplot appear like the one shown in the following screenshot.
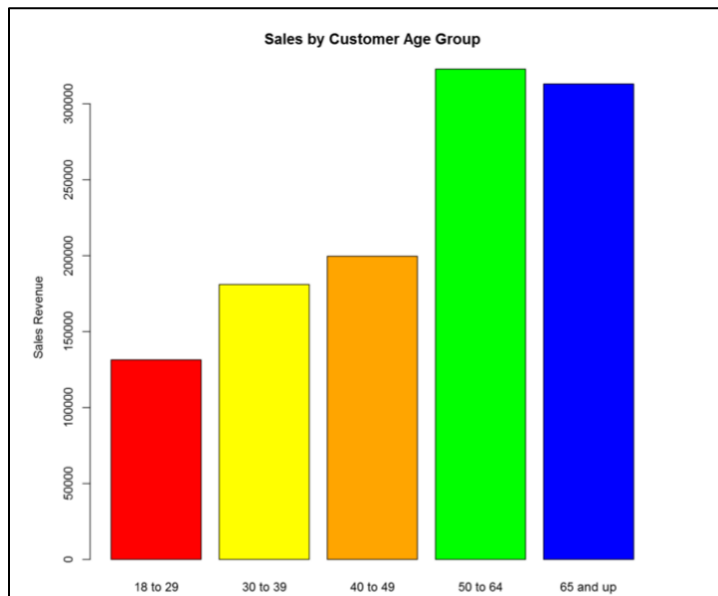
6. Add a slicer to the page to filter on state.

    a) Add a new slicer visual to the page.

    b) Add the **State** field into the **Field** well so the slicer shows a list of all states.

   c) By default, the slicer displays states in a vertical list.

    d) With the slicer visual selected, navigate to the **Format** pane and change the **Orientation** property to **Horizontal**.
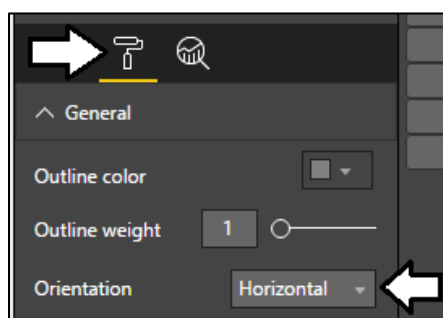
e) Adjust the **Header** property from **On** to **Off**.

f) In the **Items** section of the **Format** pane, increate the font size by adjusting the **Text** property to **14**.


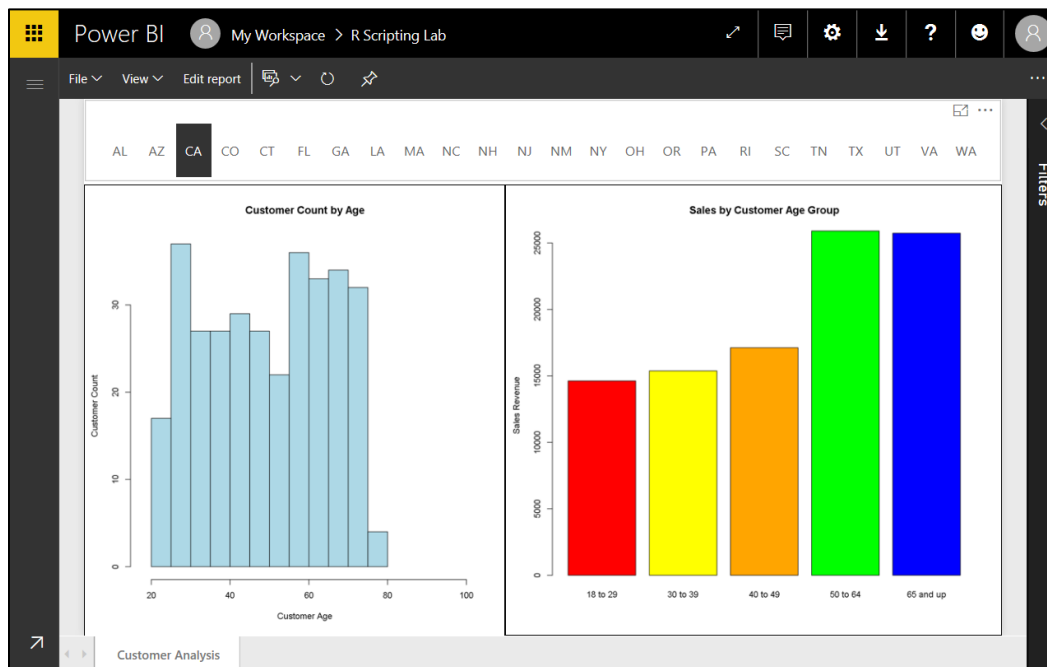g) Readjust all three visuals on the page to match the following screenshot.
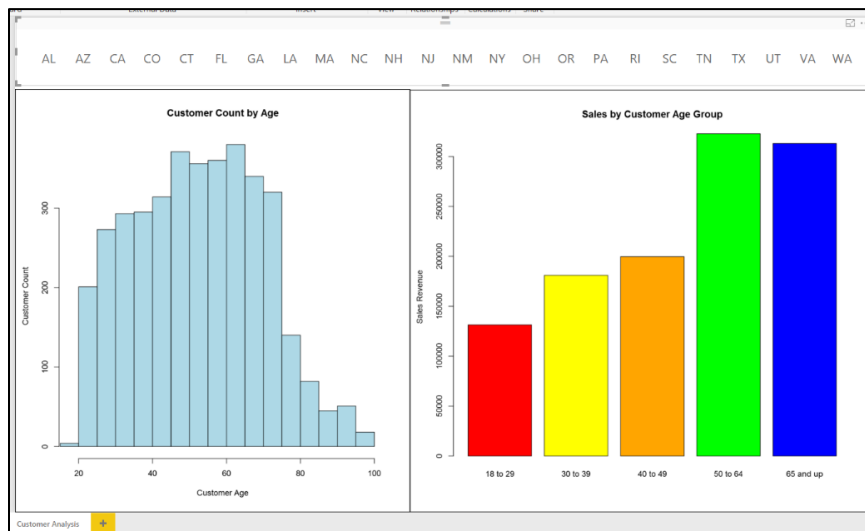
h) Test out your work. Try selecting various states from the slicer and observe how the R script visuals automatically refresh themselves with the new data from the selected state.
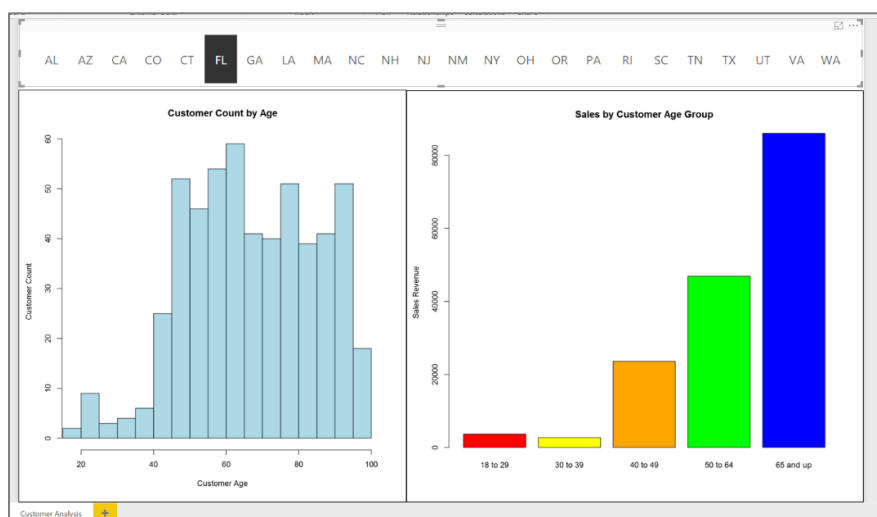
7. Save your work by clicking the Save button in the upper right-hand corner of the main window of Power BI Desktop.

8. Publish your project and test out the report in the Power BI service.

    a) Click on the **Publish** button to publish the **R Scripting Lab.pbix** project to your personal workspace.
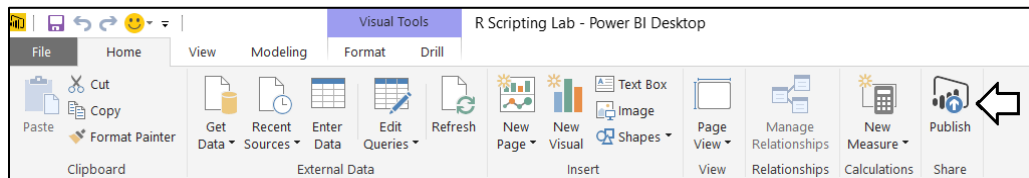
    b) Once the project has been published, test out the report you have just designed using the browser.

You have now completed this lab.