

# Getting Started With R and RStudio

## Introduction

We will be using R and RStudio to do many of our statistical calculations. First of all, we assume that you have never used R or RStudio before, so we will start from scratch! R is a free statistical software package used all over the world by lots of people at universities and workplaces that you can download onto your computer. RStudio provides the "front page" for you to use R. As you can see from the recent magazine cover below, R and RStudio are becoming the go to packages for data science.



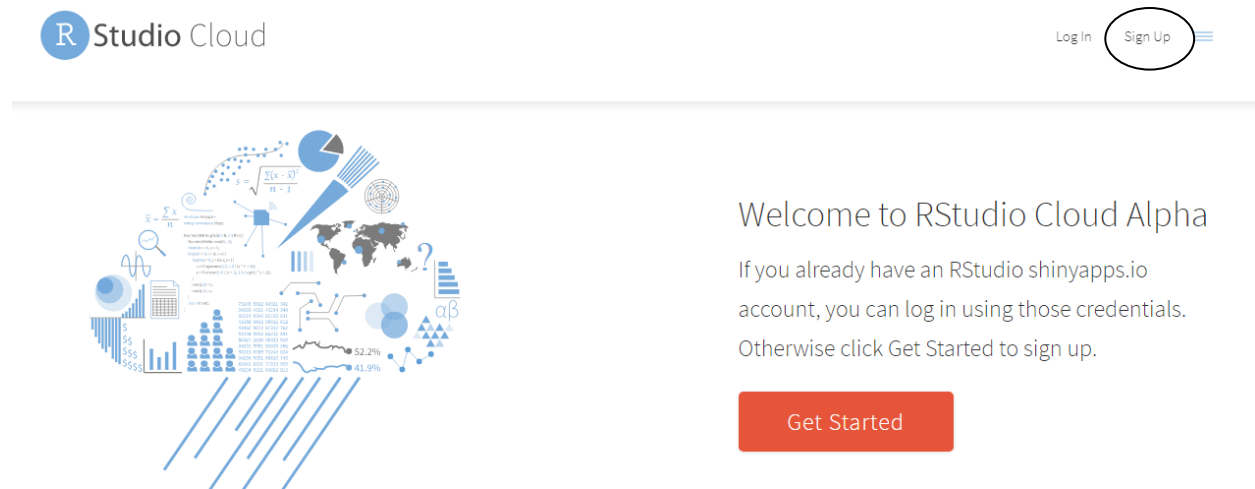
## Installing R and R Studio (or not)

One can either work locally (on your own machine) by downloading and installing R and RStudio, or work in the cloud accessing RStudio from a web browser. You are welcome to do either but for the computer phobic working in the cloud from a web browser is the easiest way to go.

For advanced students who really want R on their machine you can go to <http://cran.us.r-project.org/> to download R and then <https://www.rstudio.com/products/rstudio/download/> to download RStudio. More details are available on the course web site.

## Accessing RStudio in a Web Browser

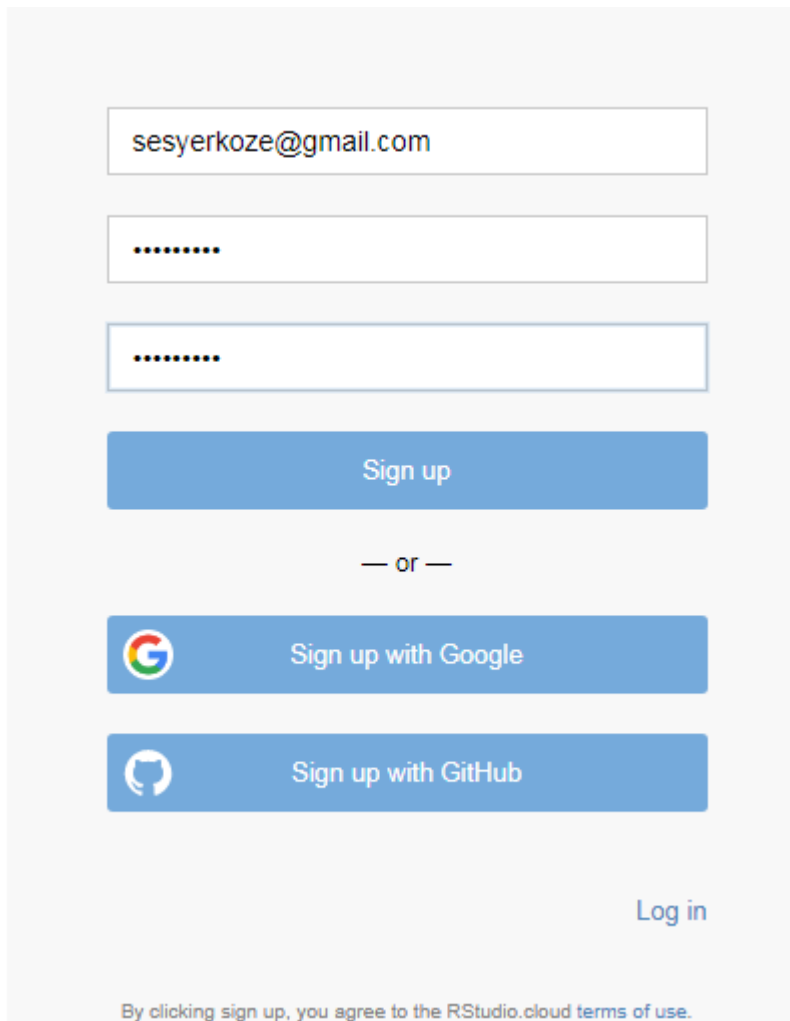
In a web browser, go to [rstudio.cloud](https://rstudio.cloud). The following will be displayed:



Choose Sign Up from the upper right hand corner to create an account:

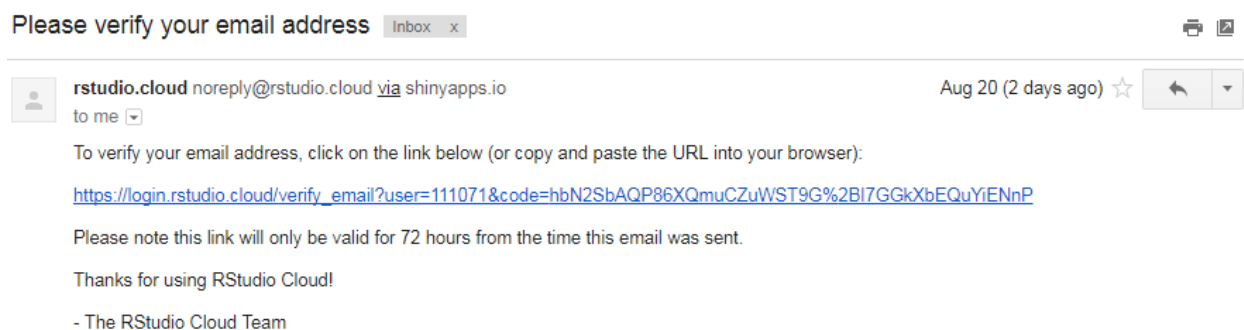
The image shows the RStudio sign-up form. At the top left is the 'R Studio' logo. The form contains the following elements: an 'Email Address' input field, a 'Choose Password' input field, a 'Confirm Password' input field, a blue 'Sign up' button, a separator '— or —', a button with the Google logo and text 'Sign up with Google', a button with the GitHub logo and text 'Sign up with GitHub', a 'Log in' link at the bottom right, and a footer note: 'By clicking sign up, you agree to the RStudio.cloud terms of use.'

Use any email you want to create an account (Here I am creating an account for Sesyer Koze, the famous Eastern European Statistician).

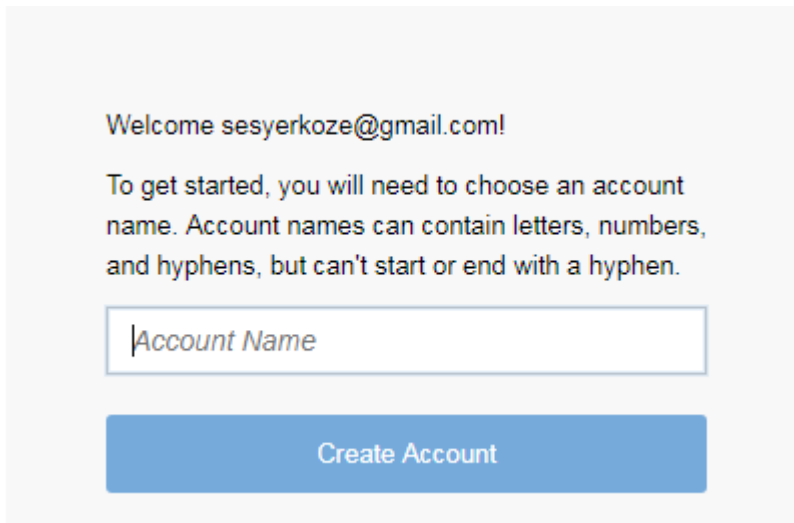


The image shows a web form for creating an RStudio Cloud account. It features three input fields: the first contains the email 'sesyerkoze@gmail.com', the second and third are masked with dots. Below these is a blue 'Sign up' button. A separator '— or —' is followed by two more blue buttons: 'Sign up with Google' (with the Google logo) and 'Sign up with GitHub' (with the GitHub logo). A 'Log in' link is positioned to the right. At the bottom, a small text line states: 'By clicking sign up, you agree to the RStudio.cloud terms of use.'

You will be emailed an authentication link-be sure to click on it.



Once you log in you will need an account name:

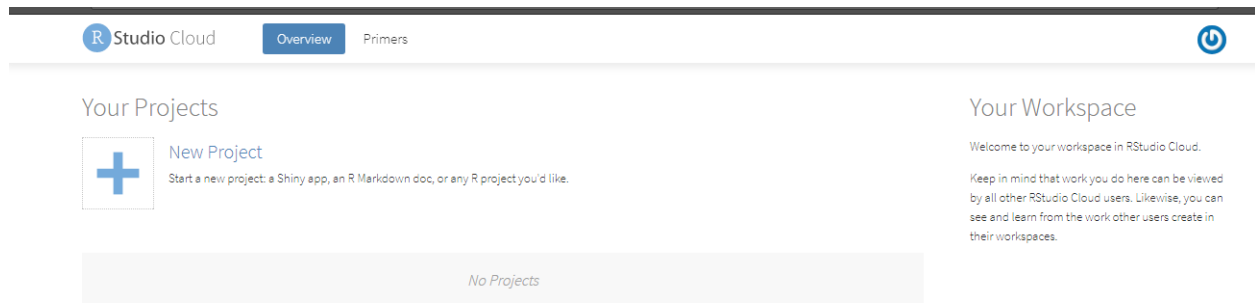


Welcome sesyerkoze@gmail.com!

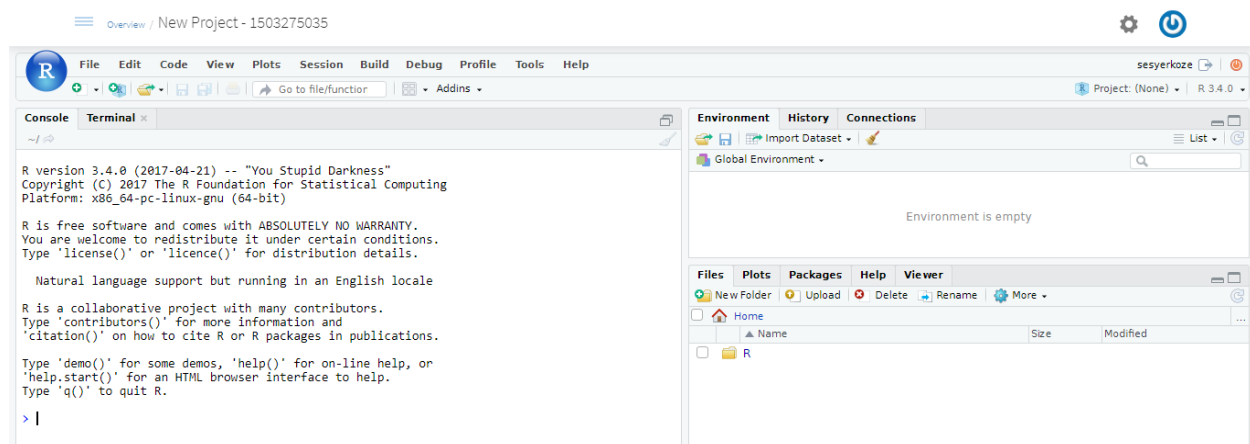
To get started, you will need to choose an account name. Account names can contain letters, numbers, and hyphens, but can't start or end with a hyphen.

Create Account

Once logged in fully you will see the following screen.



Click on the big “plus” sign, give it a few moments and you’re ready to go. Your screen should look something as follows:



## Caution

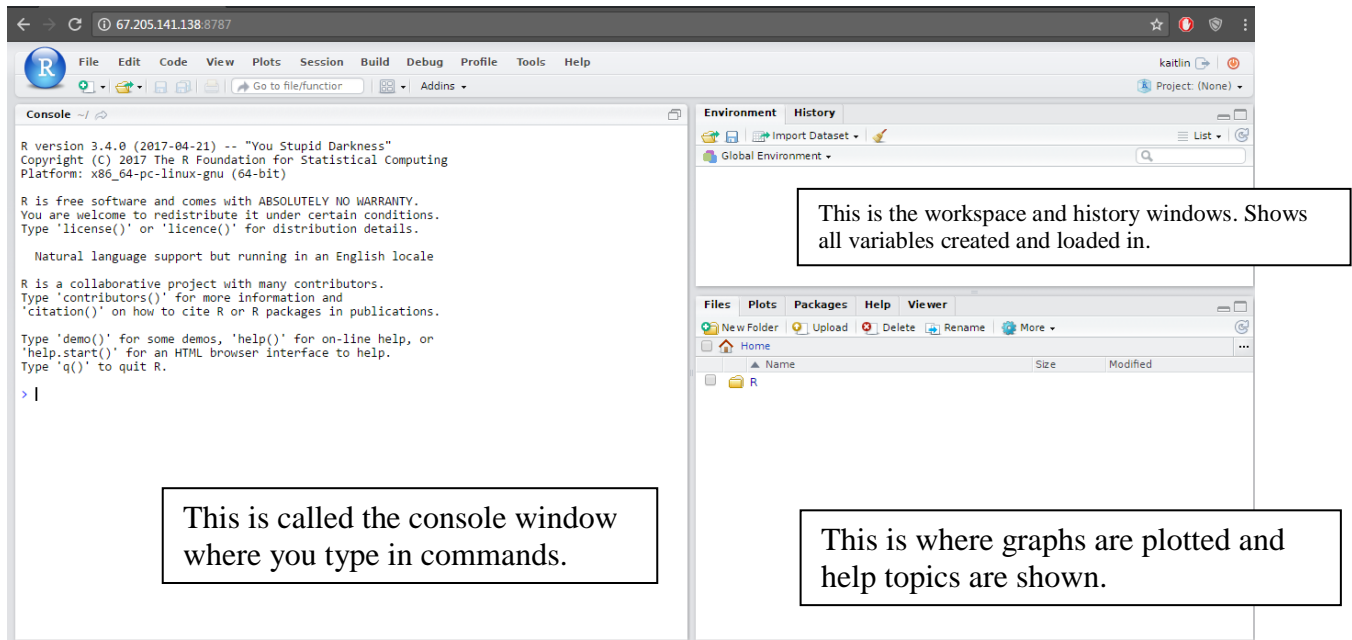
You can stay logged into Rstudio as long as you want; that is you can keep your browser window open. It will however log you out after a while without you realizing it (but your work is all stored).


If you go back to your Rstudio window after a time period, try entering commands and see a “403 error”(see below), you simply need to refresh your browser’s window to log back into Rstudio and continue where you left off.


```
Error: Status code 403 returned  
Error: Status code 403 returned  
Error: Status code 403 returned  
> |
```

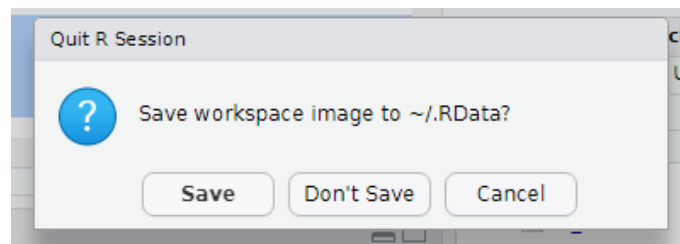
## Using R Studio for the first time

Once logged into the RStudio server you will see the following. Try not to be overwhelmed. Although R Studio is very powerful, for our intro class **we will use it mainly as a fancy graphing calculator**. In more advanced classes, we explore the software further.



To exit RStudio either close the browser window or click the power button icon  in the upper right hand of the window.

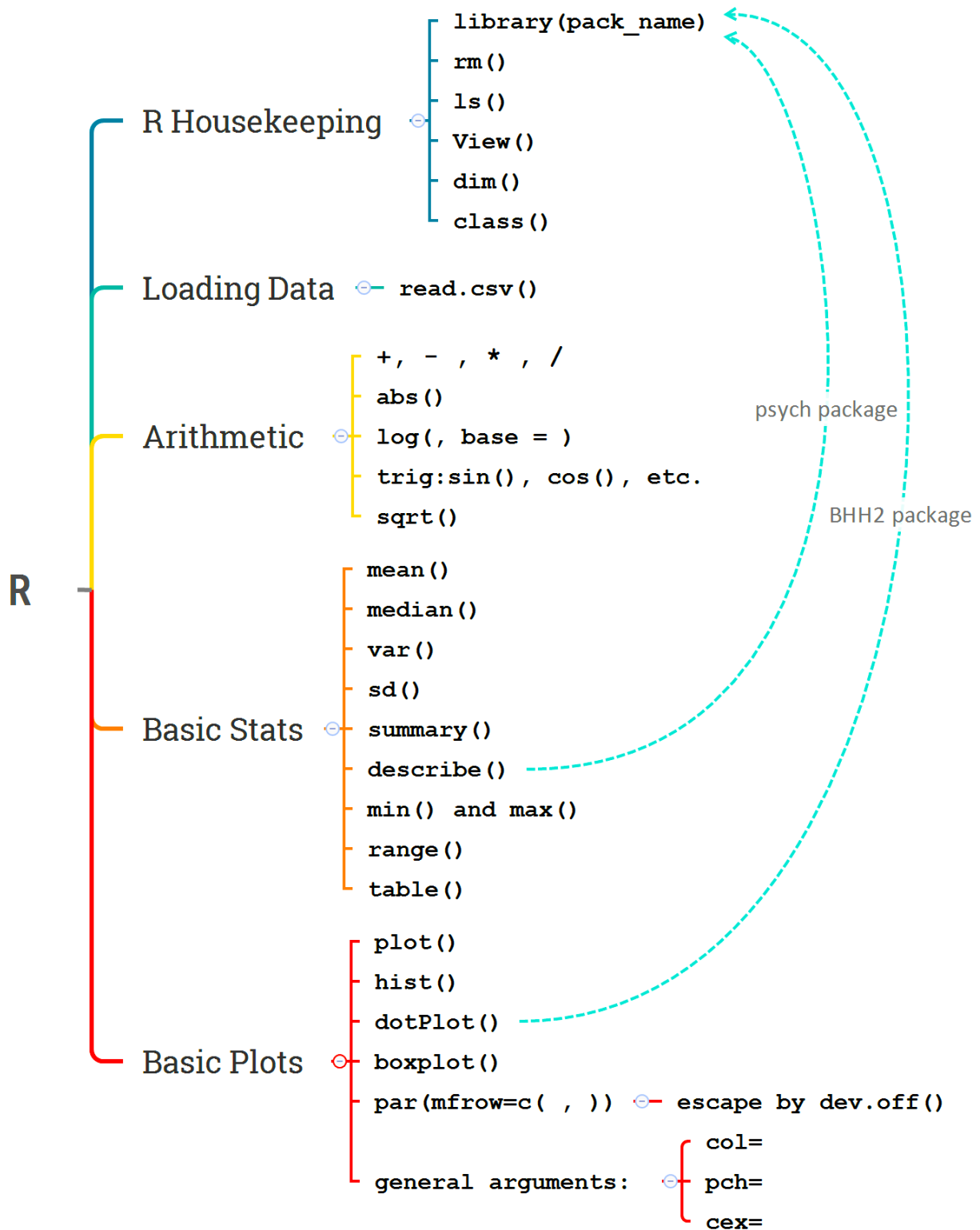
If you click the power button icon  you will be presented with the following window:



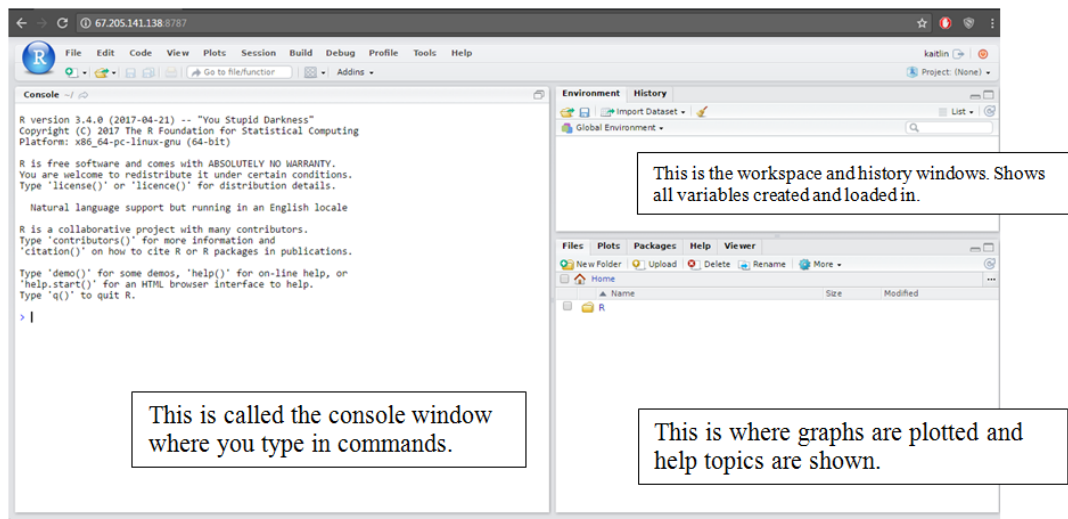
If you hit Save, everything you've done will be saved for the next time you go into Rstudio. This is generally the recommended choice.

# A First Tutorial in R

## Summary of Functions: Week 1



**Note-in the following all the R commands are entered in the console window, the lower left window on the screen.**



## Part One: Arithmetic and Variables

Arithmetic

- + , - , \* , /
- abs()
- log(, base = )
- trig:sin(), cos(), etc.
- sqrt()

Type **8+3** and press return. It doesn't matter whether there are spaces between the values or not.

```
> 8 + 3
[1] 11
```

The answer is printed in the console as above. We'll come on to what the [1] means at the end of this section.

```
> 27 / 5
[1] 5.4
```



R does everything that a graphic calculator would do. For instance, trigonometry functions such as sine, cosine, etc. We are asking for cosine of  $-\pi$  in the following:

```
> cos(-pi)
[1] -1
```

The function **abs()** returns the absolute value and  $-2^3$  is -2 to the power of 3 which without **abs()** yields to -8.

```
> abs(-2^3)
[1] 8
```

And finally in order to take square roots we use the **sqrt()** function.

```
> sqrt(4068289)
[1] 2017
```

These calculations have just produced output in the console - no values have been saved.

To save a value, it can be assigned to a data structure name. “**=**” is generally used as the assignment operator, though “**<-**” is sometimes used as well. For now we'll use x, y and z as names of data structures, though more informative names can be used as discussed later in this section.

```
> x = 8 + 3
```

If R has performed the command successfully, you will not see any output, as the value of  $8 + 3$  has been saved to the data structure called **x**. You can access and use this data structure at any time and can print the value of **x** into the console.

```
> x
[1] 11
```

Create another data structure called **y**.

```
> y = 3
```

Now that values have been assigned to **x** and **y** they can be used in calculations.

```
> x + y
[1] 14
> x * y
[1] 33
> z = x * y
> z
[1] 33
```

Important: R is case sensitive so z and Z are not the same. If you try to print the value of Z out into the console an error will be returned as Z has not been used so far in this session.

```
> Z
Error: object 'Z' not found
```

To check what data structures you have created, enter `ls()` into the console or look at the 'workspace' tab in RStudio (the upper right window).

```
> ls()  
[1] "x" "y" "z"
```

We can delete an object from our environment by `rm(objname)` function.

```
> rm(y)  
  
> ls()  
[1] "x" "z"
```

In order to clear all the objects from the environment, we can clear the workspace by going to Session > Clear Workspace... > check includes hidden objects > Yes. And done!

**Caution:** If you use the same data structure name as one that you have previously used, then R will overwrite the previous information with the new information without warning the user.

## Example Two: Data Vector and Loading Packages, and Basic Stats

Suppose we have a small dataset we want to find the summary statistics for, i.e. the mean, median, standard deviation, variance and quartiles.

Let's say this is our data: -3 2 0 1.5 4 1 3 9

Since it is a very small set of data, we can type it in using the “c” command, short for *combine* or *concatenate*:

```
# make a variable called "ourdata" containing a vector of numbers
> ourdata = c(-3,2,0,1.5,4,1,3,8)

#what is the length of our data?
> length(ourdata)
[1] 8

# call the fifth element in ourdata
> ourdata[5]
[1] 4

# calculate mean
> mean(ourdata)
[1] 2.0625

# calculate median
> median(ourdata)
[1] 1.75

# calculate range we can also get min() and max() separately
> range(ourdata)
[1] -3 8

# calculate variance
> var(ourdata)
[1] 10.17411

# calculate standard deviation
> sd(ourdata)
[1] 3.189688

> # using summary function to get basic stats
> summary(ourdata)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-3.000  0.750   1.750   2.062   3.250   8.000
```

The functionality of R can also be extended by many different packages. We show you an extensive function below from the **psych** package for descriptive statistics. For example here we would like to use the **describe()** function part of “psych” library. So, first we need to install the package, then load the package with the **library()** function, and then we can use

the function of interest in this case, `describe()`. Note that the `install.packages` command returns a lot of screen output that can be ignored. This only has to be done once-every time you go back into Rstudio you will just have to use the `library(psych)` command to load the library- you wont have to install it again.

```
> install.packages("psych")
> library(psych)
> describe(ourdata)
  vars n mean  sd median trimmed  mad min max range skew kurtosis   se
X1    1  8 2.06 3.19   1.75    2.06 2.22  -3   8   11  0.3    -0.66 1.13
```

### Example Three: Load Data from File, `attach()`

Sometimes the dataset is much larger and we prefer to read it in directly into RStudio, rather than type it in by hand as above in Example Two.

We have an old data set on cars from 1978, accessible at <http://people.fas.harvard.edu/~mparzen/stat104/cars10.csv>


This data set may be read into RStudio as follows (and this is how we will read in all data sets for this class).

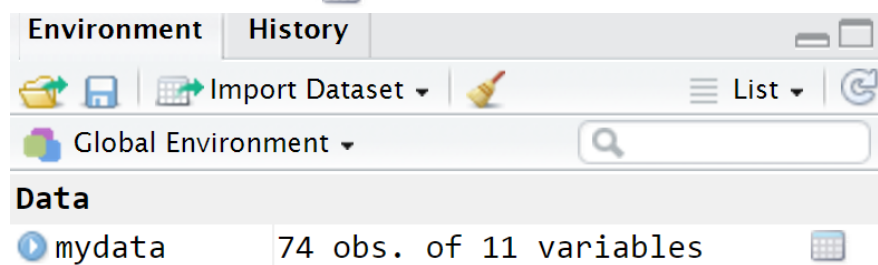
```
#loading the data with read.csv() command
> mydata=read.csv("http://people.fas.harvard.edu/~mparzen/stat104/cars10.csv")

# getting the name of columns
> names(mydata)
[1] "make"      "price"      "mpg"        "headroom"   "trunk"
"weight"
[7] "length"    "turn"       "displacement" "gear_ratio" "foreign"
```

The `dim()` command tells us the *dimensions* of our data set; we have 74 rows of data on 11 variables.

```
> dim(mydata)
[1] 74 11
```

Sometimes it helps to visualize the data in a tabular, excel-like format. To do so, we can use the function `View()` or just simple  click on the table icon by our data frame variable.



## Accessing data or a variable

We can access our data in several ways. The obvious but cumbersome approach is as follows:

- 1) The “\$” is the method to work with different variables in the data frame.

```
> mydata$mpg
[1] 22 17 22 20 15 18 26 20 16 19 14 14 21 29 16 22 22 24 19 30 18 16 17 28
[25] 21 12 12 14 22 14 15 18 14 20 21 19 19 18 19 24 16 28 34 25 26 18 18 18
[49] 19 19 19 24 17 23 25 23 35 24 21 21 25 28 30 14 26 35 18 31 18 23 41 25
[73] 25 17
```

Now, it seems appropriate to talk about the meaning of number in the brackets. By the `dim()` function we learned that the car data that now we have saved into `mydata` variable has 74 rows and 11 columns.

```
> dim(mydata)
[1] 74 11
```

When we use the “\$” operator, R returns a *list* of mpg data and the number in the brackets corresponds to location of data points in the list. For instance in the last line we see, `[73] 25 17`, which means 73<sup>rd</sup> miles per gallon (mpg) is 25 and therefore 74<sup>th</sup> is 17. It’s just an R convention: dataframes (2D) have a dimension of rows x columns, and we call on a column name with \$ operator, we get a 1-dimensional list. You might want to verify this by `View(mydata)` and look at the 73<sup>rd</sup> and 74<sup>th</sup> data point in the mpg column.

Now that we have learned to extract a column of data from a data frame. We can call the functions on the selected column. For example, if we want to know the median value of mile per gallon of cars we run the following:

```
> median(mydata$mpg)
[1] 20
```

- 2) Alternatively we can “attach” the data set into RStudio’s memory and work directly with the variable names as follows:

```
> attach(mydata)
> median(mpg)
[1] 20
```

**Caution:** It is good practice to detach the data frame after being done.

```
> attach(mydata)
> sd(mpg)
[1] 5.785503
```

# it is very important to detach() especially while working with multiple dataframes.

```
> detach(mydata)
> sd(mpg)
Error in is.data.frame(x) : object 'mpg' not found
```

```
# we still can use the $ to access the mpg column
```

```
> sd(mydata$mpg)
```

```
[1] 5.785503
```

Eventually, we will learn shortcuts such as follows to look at the descriptive statistics for three variables at once.

```
> describe(mydata[,c("mpg","price", "length")])
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
mpg	1	74	21.30	5.79	20.0	20.77	5.19	12	41	29	0.93	0.87	0.67
price	2	74	6165.26	2949.50	5006.5	5614.17	1358.06	3291	15906	12615	1.62	1.69	342.87
length	3	74	187.93	22.27	192.5	188.03	28.17	142	233	91	-0.04	-1.01	2.59

In the Appendix we learn how to upload our own data or data of our interest into the web-based RStudio.

## Data Visualization: Show me the data!



We will cover two subjects in the second part of this introduction to R: 1) How to make plots, and 2) how to export the plots from RStudio.

## Basic Plots

```
plot()
hist()
dotPlot()
boxplot()
par(mfrow=c( , ))  escape by dev.off()

general arguments: { col=
                    pch=
                    cex=
```

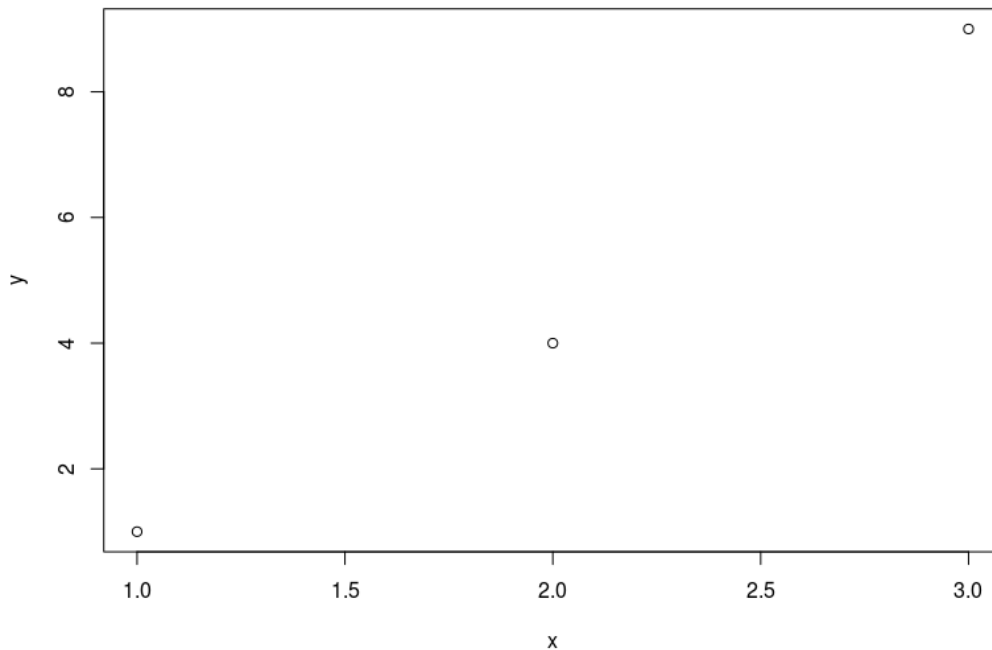
## Plotting Data Points: `plot()`

Let's define x and y as below:

```
> x <- c(1,2,3)
> y <- c(1,4,9)
```

The **`plot()`** function needs two sets of data points and will plot them against each other. Then, I will introduce some of the options we can use to make our plots more detailed, informative, and representative.

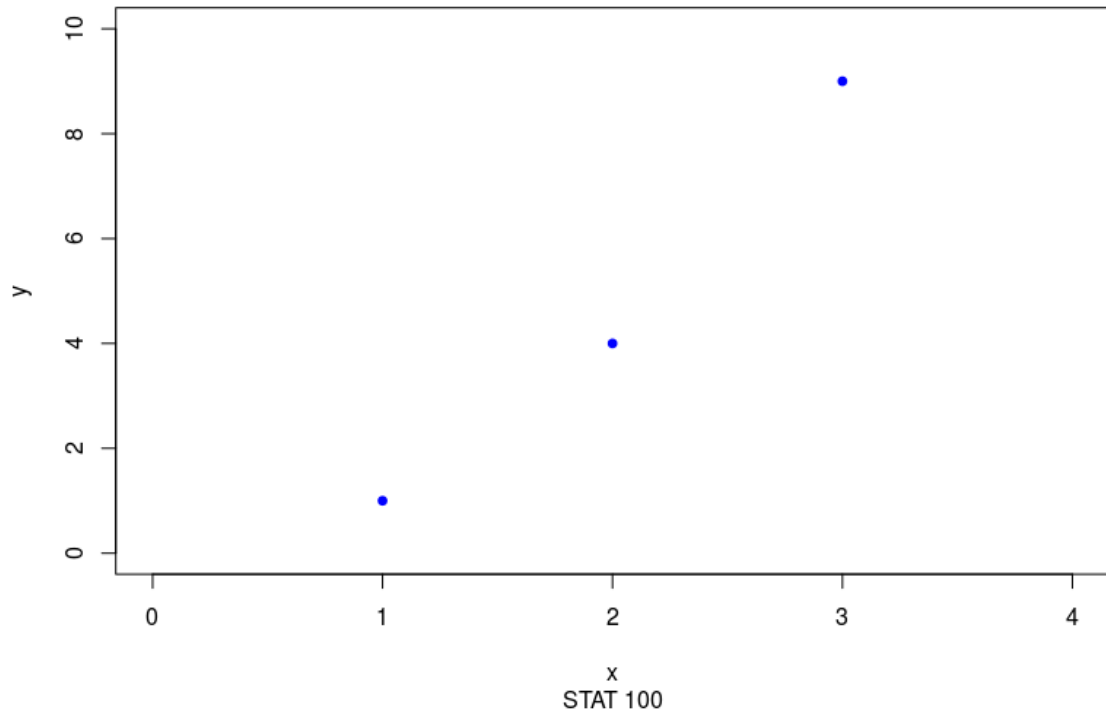
```
> plot(x,y)
```



```
> plot(x,y, xlab = "x", ylab="y", pch = 19, cex=0.8, col = "blue", xlim = c(0,4),
ylim=c(0,10), main="Our First Plot!",sub = "STAT 100")
```



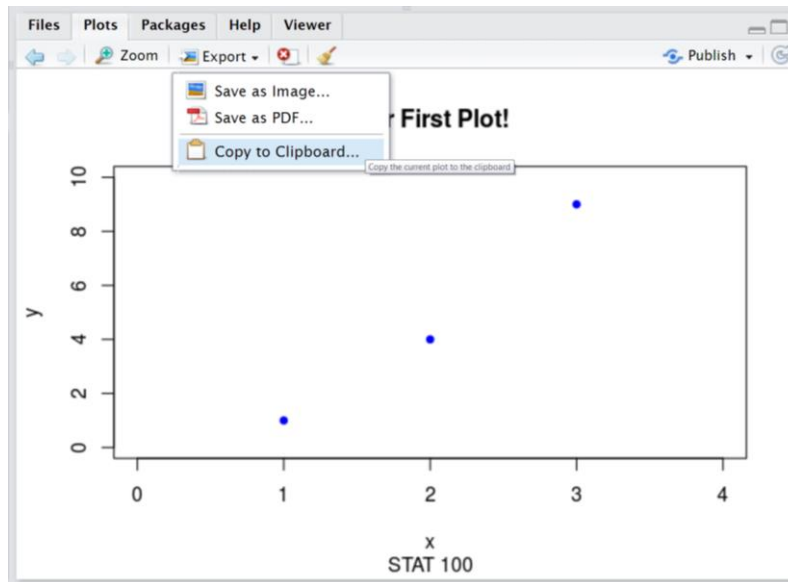
**Our First Plot!**



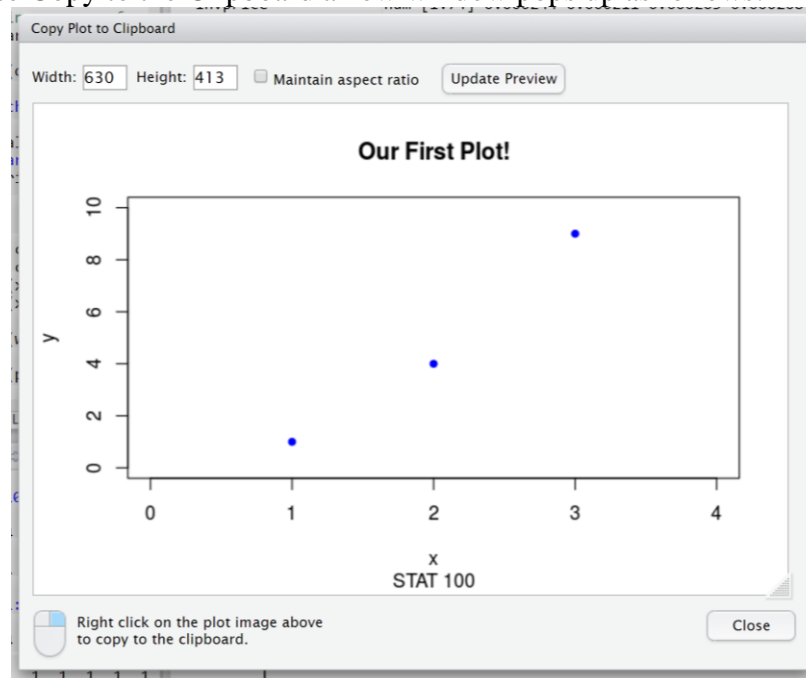
## Copying Plots into Microsoft Word

Now, how do we get this histogram into Word (or OpenOffice) for a nice homework submission?

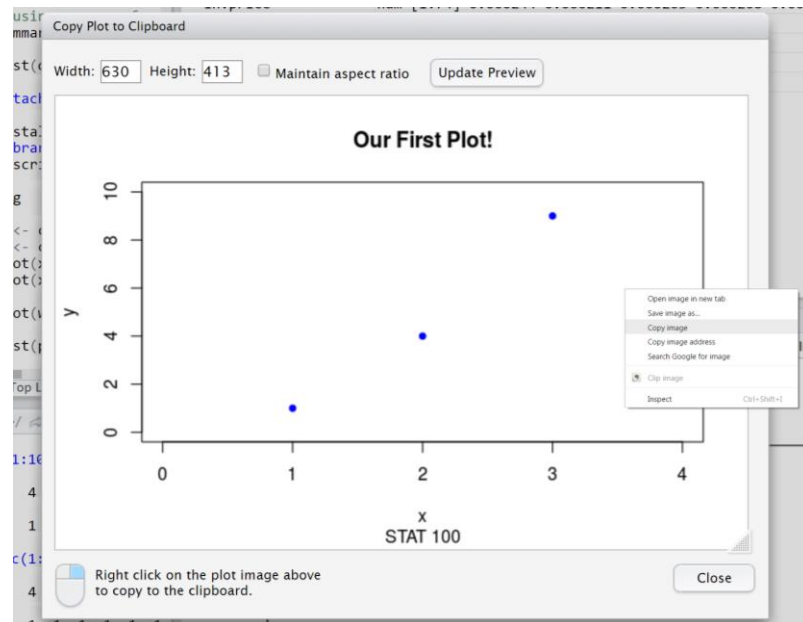
In the graphics window, go to the Export option and choose “Copy to Clipboard”.



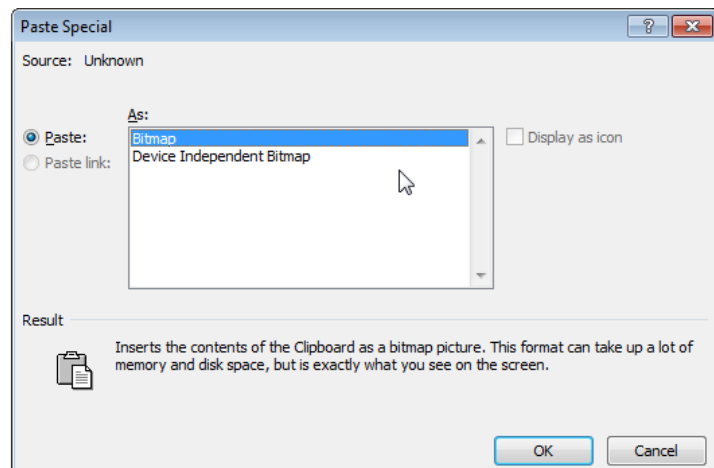
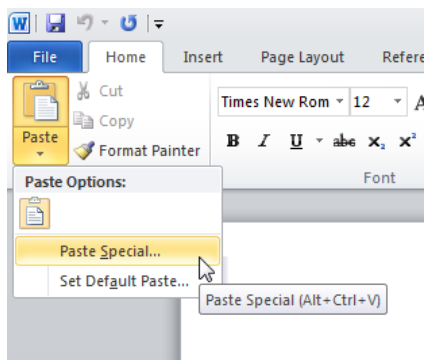
When you choose Copy to the Clipboard a new window pops up as follows:



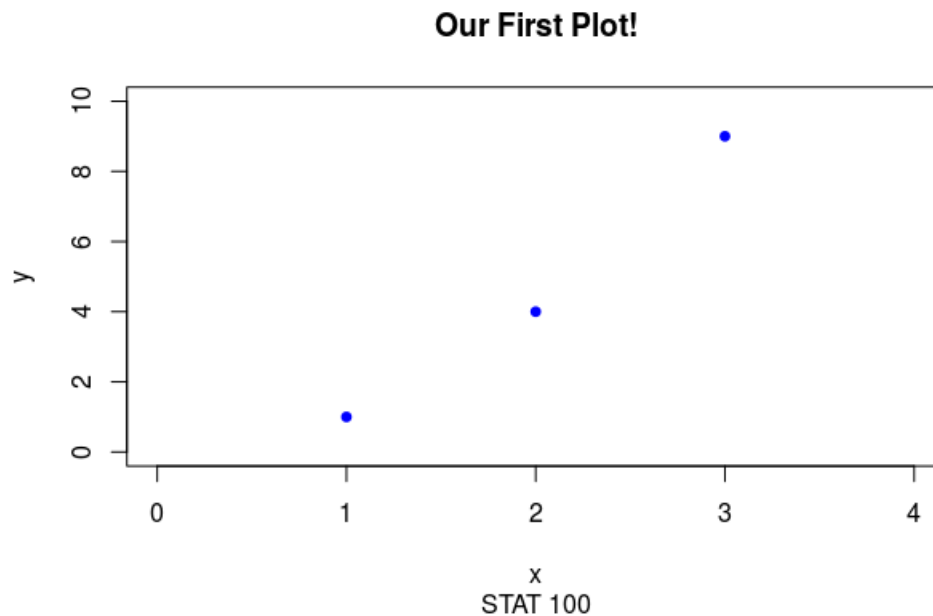
On a pc, you need to right-click on this window (or hold down the control and click on a mac, or use two fingers on the touchpad) and choose “copy image”.



Once the image is copied to the clipboard, you need to use “Paste Special” in Word to paste it as a bitmap (in Word on a PC) or an image (in Word on a Mac).

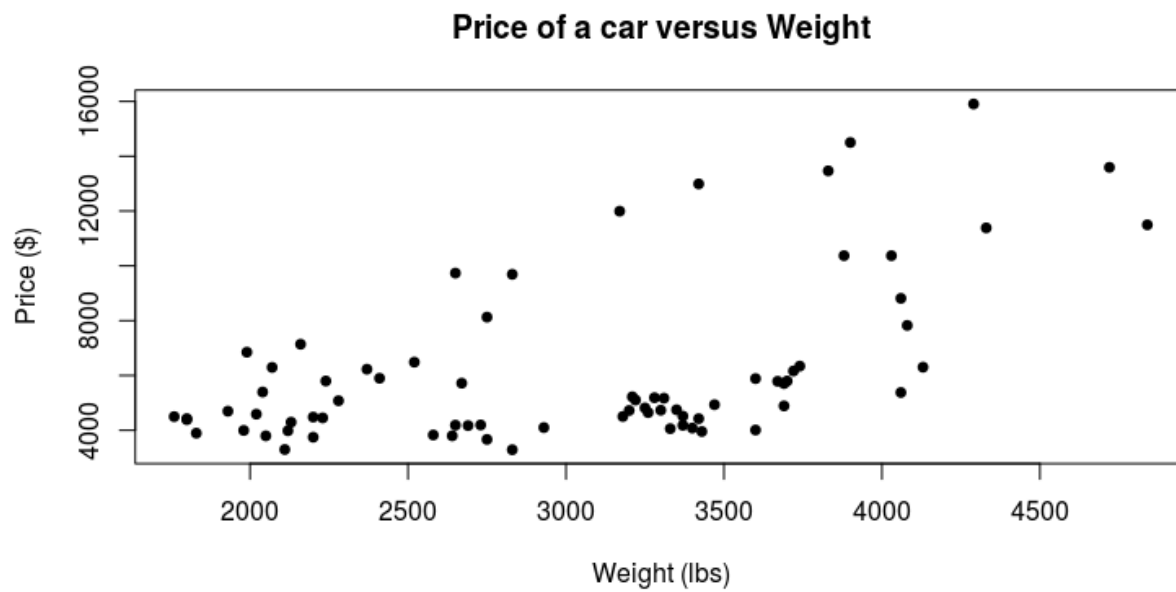


Once it is pasted it should look as follows, and can be moved around and re-sized.



Now, let make a scatterplot of weight and price of the car data. We put the essential price and weight in to the plot function and rest is just to make the plot prettier. The `pch` argument defines the shape of the dots. In Appendix 3, you will find all the shapes we can use, `pch = 19` gives us filled dots.

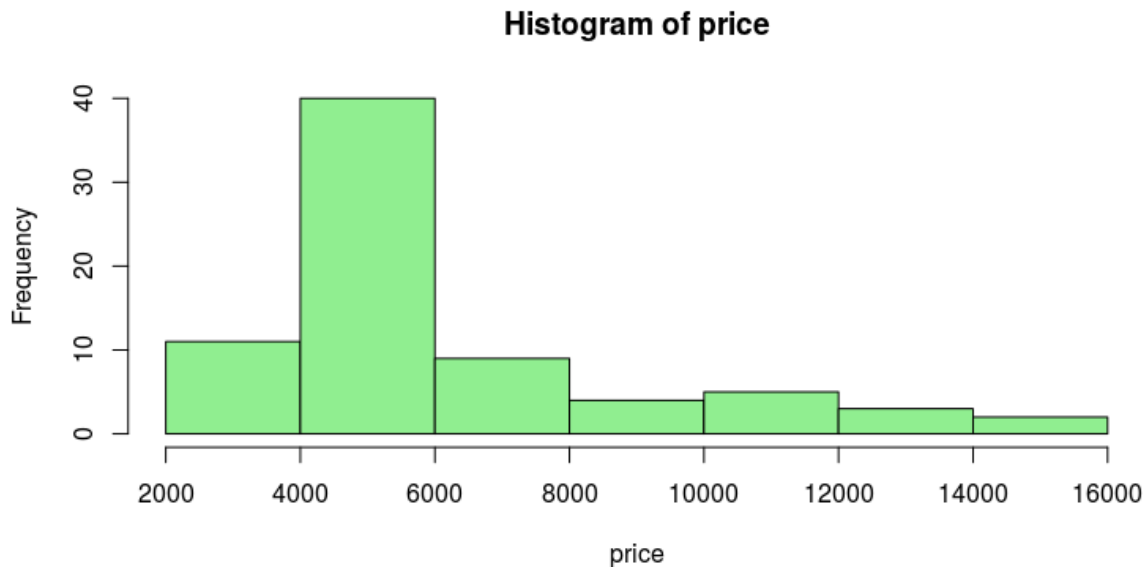
```
> plot(weight,price,main="Price of a car versus Weight", pch = 19, cex=
0.8,xlab= "Weight (lbs)", ylab="Price ($)")
```



## Histograms: `hist()`

One can also create new variables in R, such as transformations of existing variables. Consider the variable `price`. Here is a histogram of the variable.

```
> hist(price, col = "lightgreen")
```

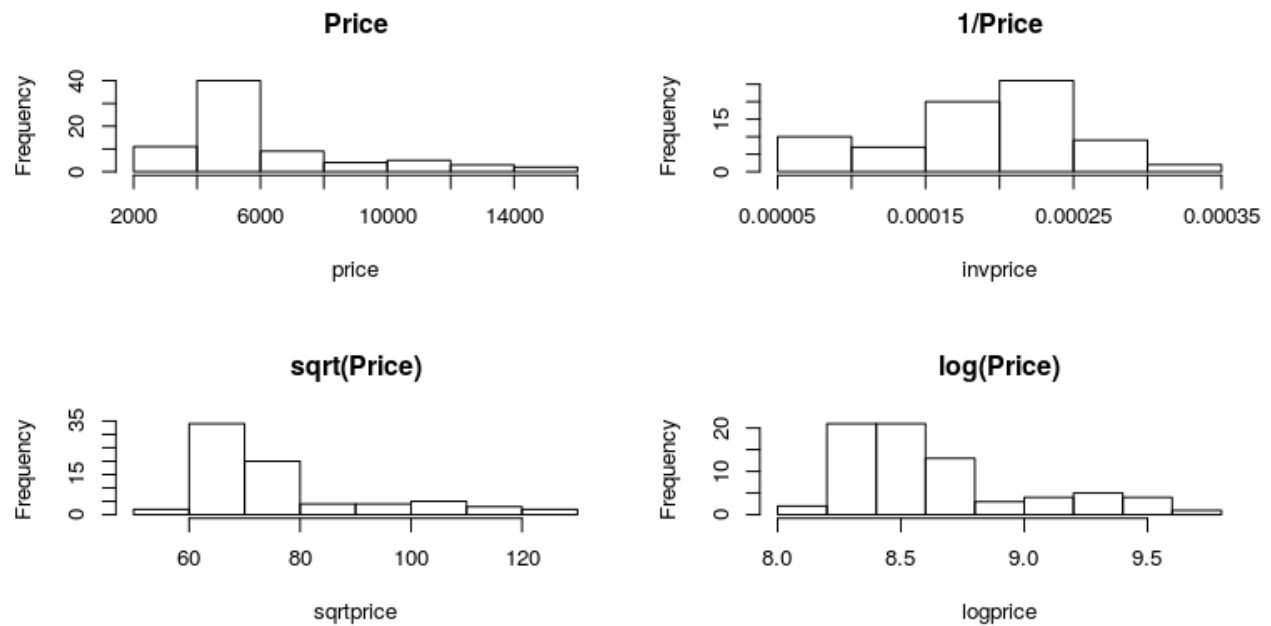


We are going to see if we can find a transformation that makes it more symmetric. The `par()` command (as in parameter) combined with `mfrow` argument (as in matrix from rows) used below lets us plot several graphs at once on the same page and compare the data more side by side.

```
# Transforming the price data
> invprice=1/price
> logprice=log(price) #simple log by default is the natural log
> sqrtprice=sqrt(price)

# ask for a 2x2 matrix pattern for plots
> par(mfrow=c(2,2))

# make histogram of price and 3 transformed price data points
> hist(price,main="Price")
> hist(invprice,main="1/Price")
> hist(sqrtprice,main="sqrt(Price)")
> hist(logprice,main="log(Price)")
```



**Tip:** If we make another plot, R still thinks we would like to continue with `par(mfrow=c(2,2))` pattern. To escape and make independent plots again, we would need this command: **`dev.off()`**

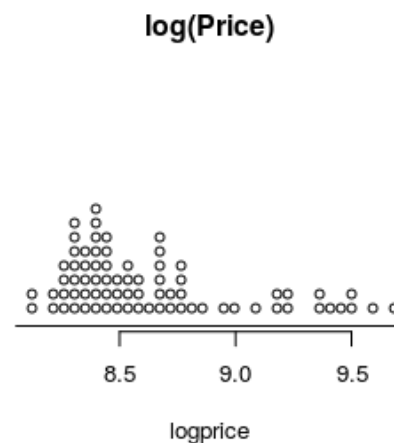
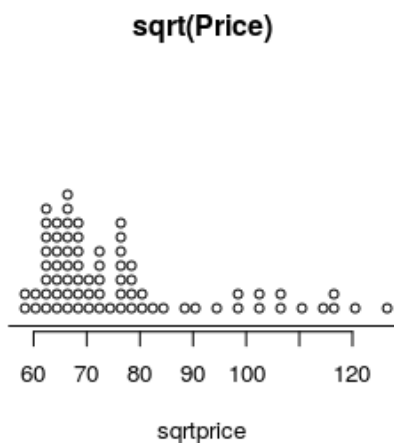
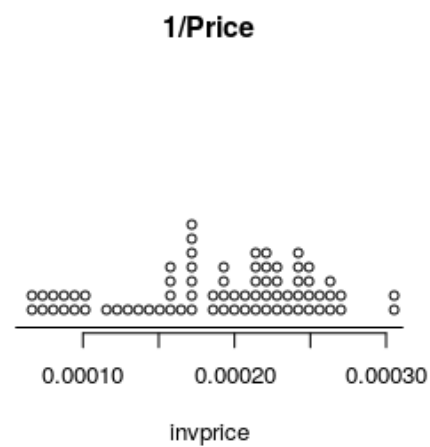
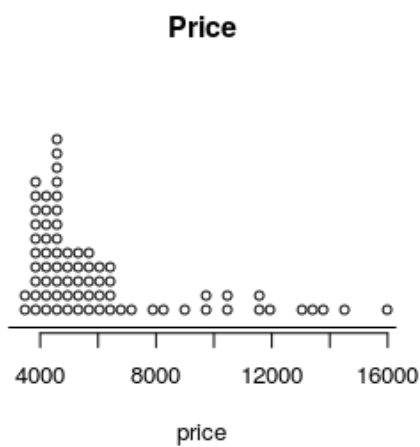
## Dot Plots: *dotPlots()*

An alternative way to visualize data points distributions can be done by the `dotPlots()` function. This function resides in BHH2 package that we will load by calling it with the `library()` function.

```
# loading BHH2 Package
> install.packages("BHH2")
> library(BHH2)

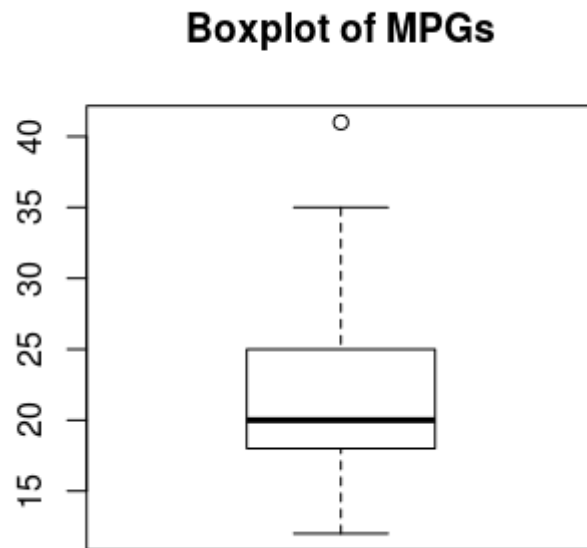
# making dot plots of price and the 3 transformed price data points
> dotPlot(price,main="Price")
> dotPlot(invprice,main="1/Price")
> dotPlot(sqrtprice,main="sqrt(Price)")
> dotPlot(logprice,main="log(Price)")

# escape from par(mfrow=c(2,2))
> dev.off()
```



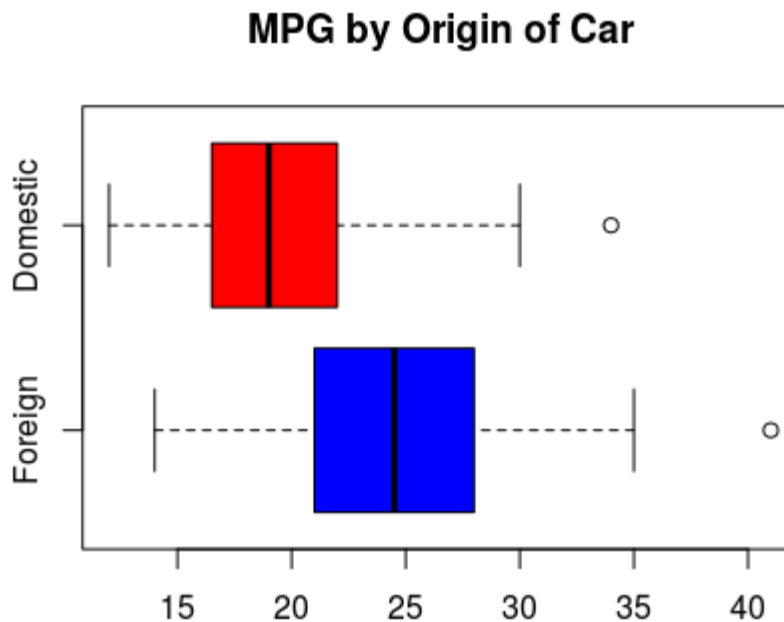
### **Boxplots: `boxplot()`**

```
> boxplot(mpg, main="Boxplot of MPGs")
```



With options to the function call one can create all sorts of plots. Here is a side by side boxplot.

```
> boxplot(mpg[foreign=="Foreign"],mpg[foreign=="Domestic"],horizontal=TRUE,names=c("Foreign","Domestic"),main="MPG by Origin of Car", col = c("blue","red"))
```





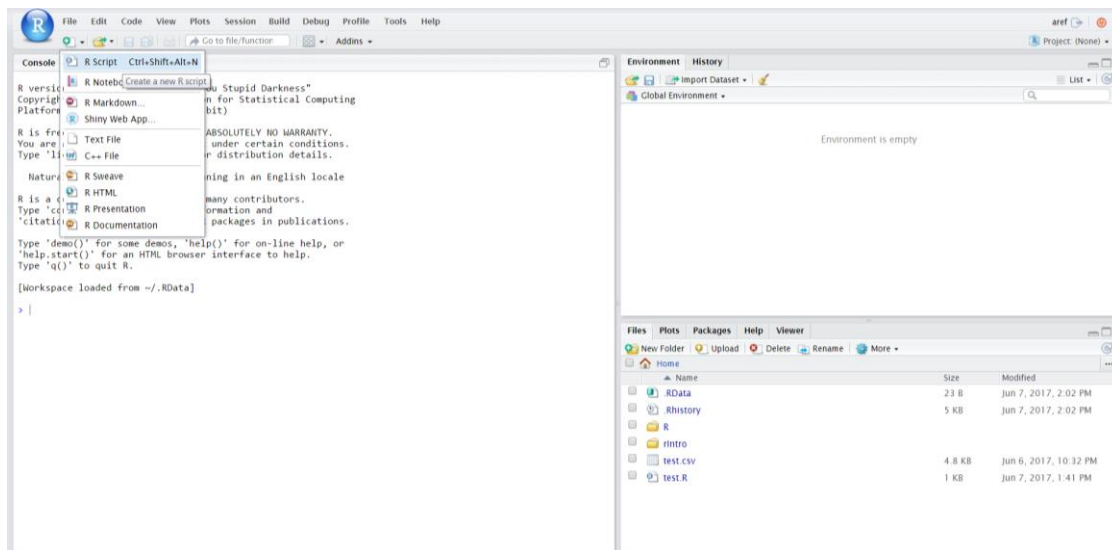
## Brief Summary of Commands:

Command	Function	Example
Load Data		
<code>read.csv()</code>	reads a csv file in to R	<code>read.csv("file_name.csv")</code>
Basic Stats		
<code>mean()</code>	computes back the mean	<code>mean(x)</code>
<code>median()</code>	computes back the median	<code>median(y)</code>
<code>var()</code>	computes the variance	<code>var(1:10)</code>
<code>sd()</code>	computes the standard deviation	<code>sd(c(1, 5, 6, 10, -7))</code>
<code>summary()</code>	gives a basic statistical summary	<code>summary(ourdata)</code>
<code>min()</code>	returns the minimum value	<code>min(ourdata)</code>
<code>max()</code>	returns the maximum value	<code>max(ourdata)</code>
<code>range()</code>	returns the range i.e. min and max	<code>range(ourdata)</code>
<code>table()</code>	makes a frequency table	<code>table(mydata\$mpg)</code>
Basic Plots		
<code>plot()</code>	plots a 2D graph from 1 or two sets of data points	<code>plot(x,y)</code>
<code>boxplot()</code>	makes a boxplot	<code>boxplot(mydata\$mpg)</code>
<code>hist()</code>	makes a histogram of a chosen data series.	<code>hist(mydata\$price)</code>
<code>dotPlot()</code>	makes stacked scatter plot similar to stem-and-leaf plots	<code>dotPlot(mydata\$price)</code>

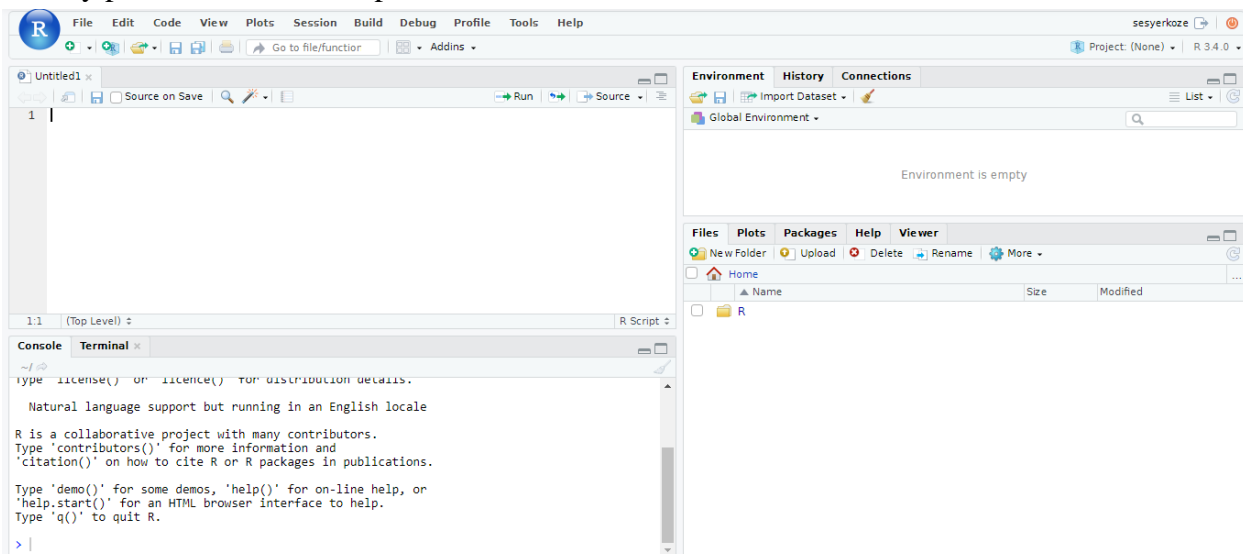
## Making an R script

Although we can just run all our codes in the console, it is considered good practice to have all the codes in the same place and that's when having a file with all the codes comes in handy for troubleshooting and reusing previous code.

To make a script, we can go to File > New File > R Script or we can click on the New icon and choose R Script as shown below.



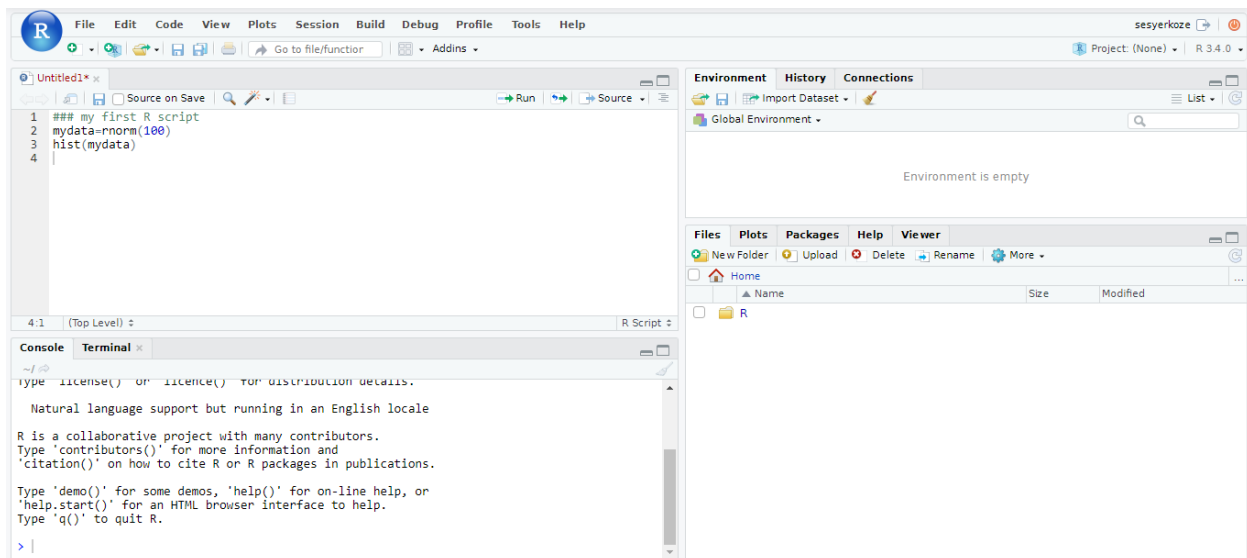
Now we have 4 windows: clock wise from top left, the script file window, Environment and History panel, File and Plots panel, and the console.



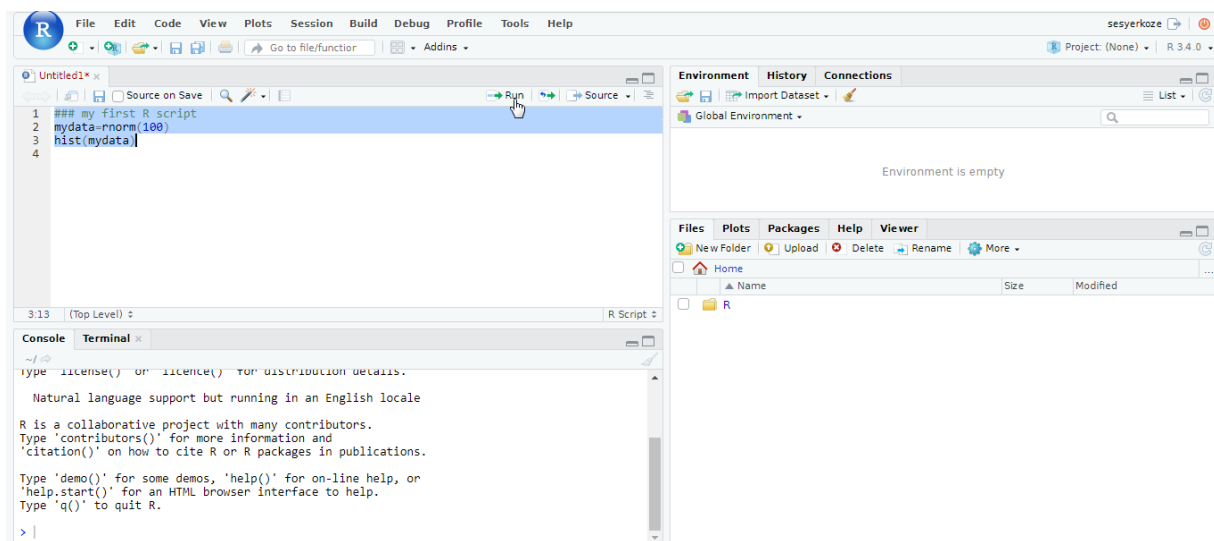
In the upper left script window we are going to enter the following three lines of R code; the first line is a comment line, the second generates some random data and the third line plots the random data.

```
### my first R script
mydata=rnorm(100)
hist(mydata)
```

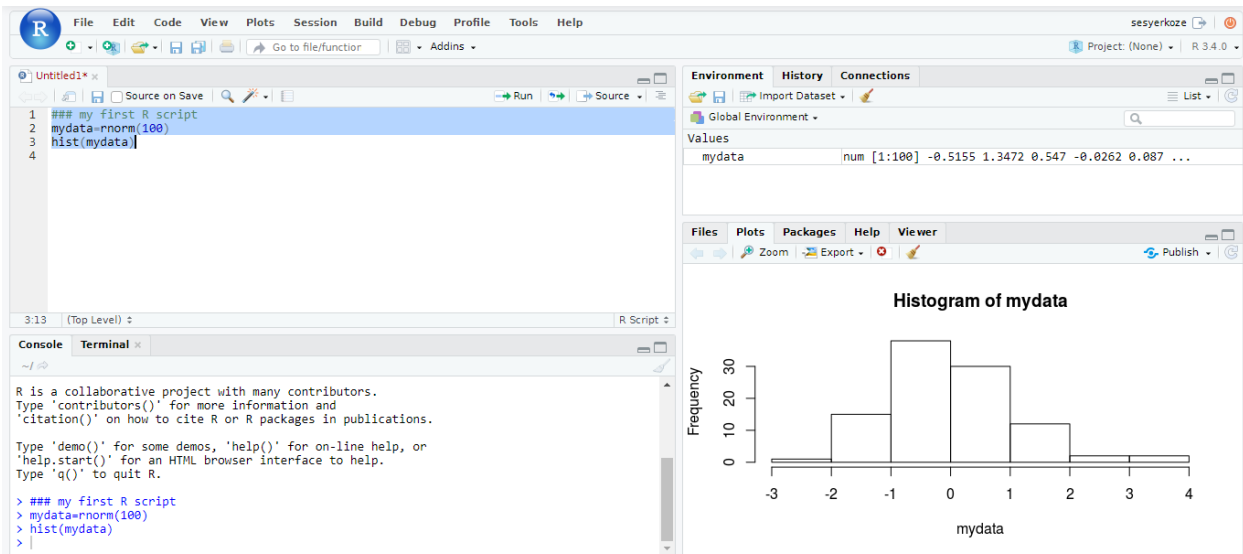
Your computer screen should then look as follows:



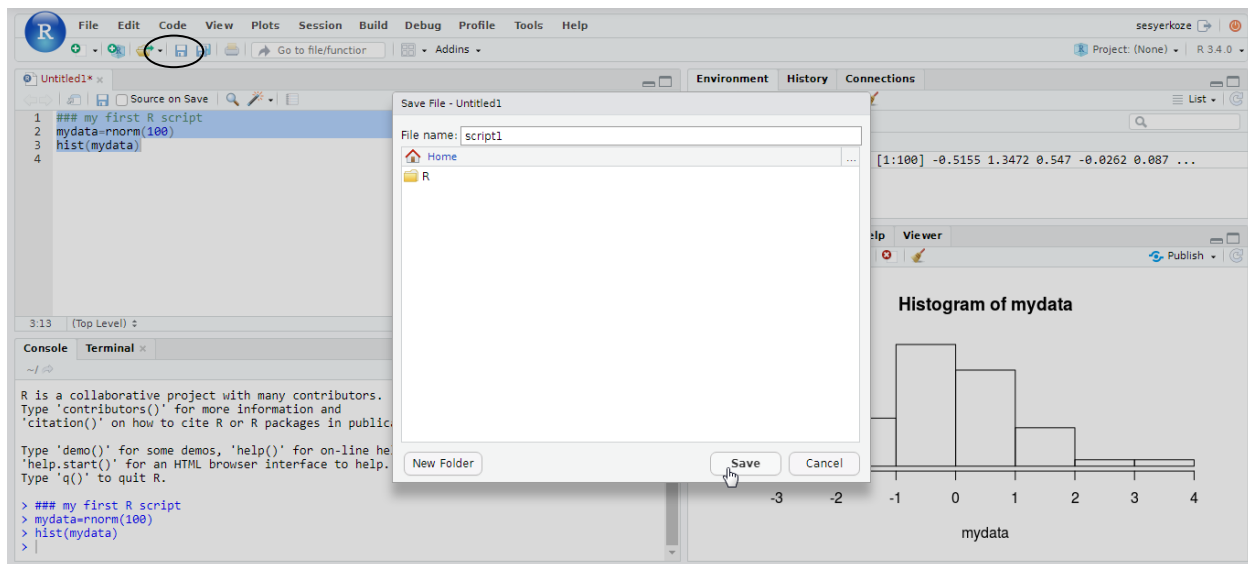
It's not enough to type in the code; we now need the computer to read and execute the code. To do so we highlight the lines of code and hit the Run button as shown below:



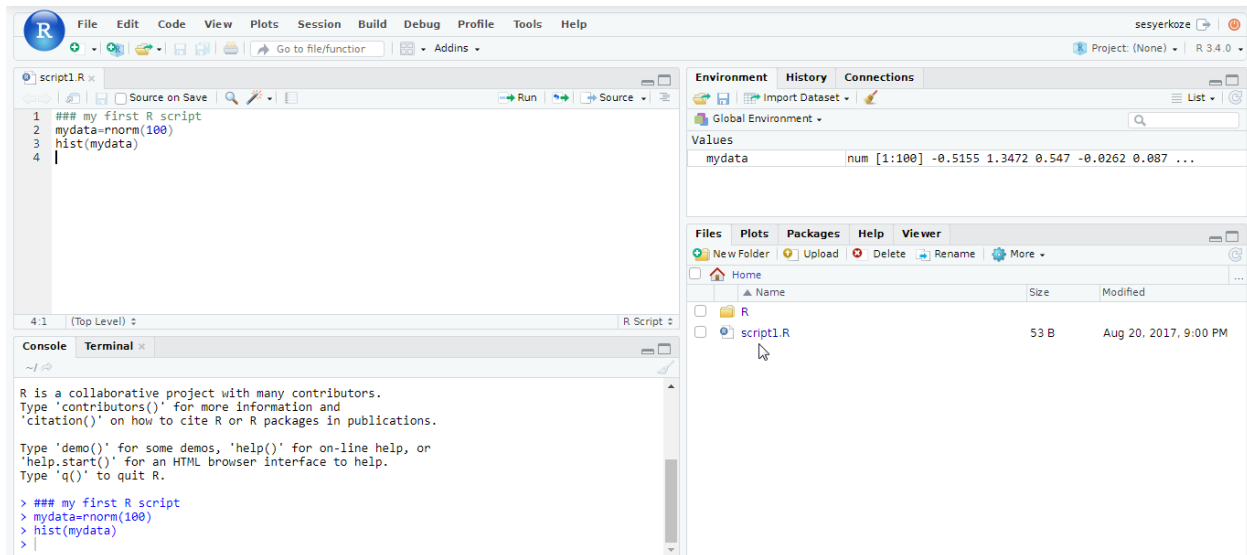
After you hit run your screen looks as follows. The only output we asked for is the histogram graph so there is nothing that useful given in the lower left console window.



To save the script we use the universal floppy disk icon for save. We need to give the script a name.



After saving the script file we can see the newly saved file in the File panel in lower right corner. If we exit and later go back into Rstudio, we can double click this filename to open it up in the script editor.



There will be a commented R Script on Canvas to accompany this tutorial. **Also, to download the R Script for the first week, run the following the command in the console:**

```
download.file("http://people.fas.harvard.edu/~mparzen/stat104/R_intro.R", "R_intro.R")
```

Clicking on the downloaded R\_intro.R script file will open the R script on the top left window.

## Appendix 1: R help

There are a lot of different ways to get help when you are using R. R has help pages that are very useful once you have familiarized yourself with the layout. Information about a function (for example `read.table`) can be accessed by typing the following into the console:

```
help(read.table)
```

or

```
?read.table
```

This should include information about parameters that can be passed to the function, and at the bottom of the page should be examples that you can run which can be very useful.

If you don't know the function name that you're after, eg. for finding out the standard deviation, try

```
help.search("deviation")
```

or

```
??deviation
```

And you can always try searching the internet but remember that 'R' in a general search isn't always very good at returning relevant information so try and include as much information as possible.

We will include some R resources on our class website. R has become very popular so there are lots of resources out there.

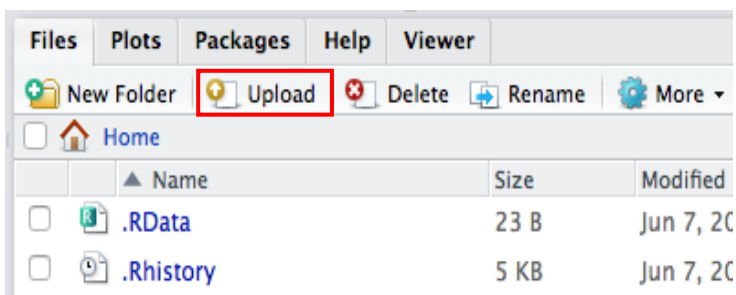
Or go to <http://www.rseek.org/> which will return more R specific information.

## Appendix 2: Loading (our own) Data into R

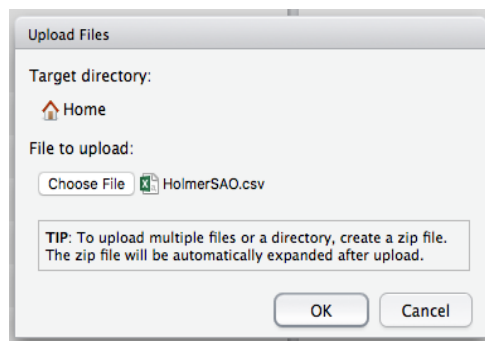
Ultimately, we would like to load our very own data—or the data that is of our interest—in R and analyze it. In this appendix, we briefly will go through uploading a CSV file (comma-separated values) into STAT 104 web-based RStudio and then load it into the R environment.

In this example, I am trying to upload the file “HolmerSAO.csv” to our R environment. This dataset has the number of surgeons, anesthesiologists, and obstetricians in the surveyed countries.

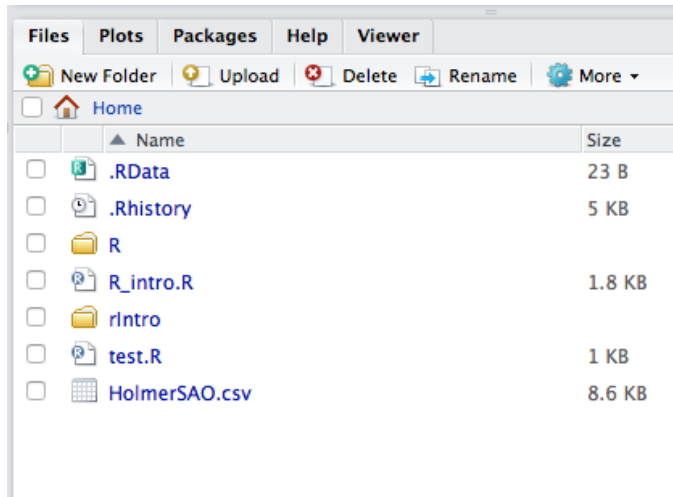
1- In Files session, click on Upload:



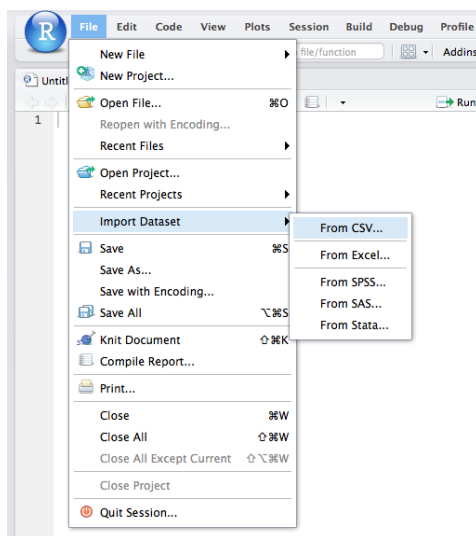
2- Choosing the file want to upload.



3- Now we have the file in Home directory. Let's load it to our R environment.

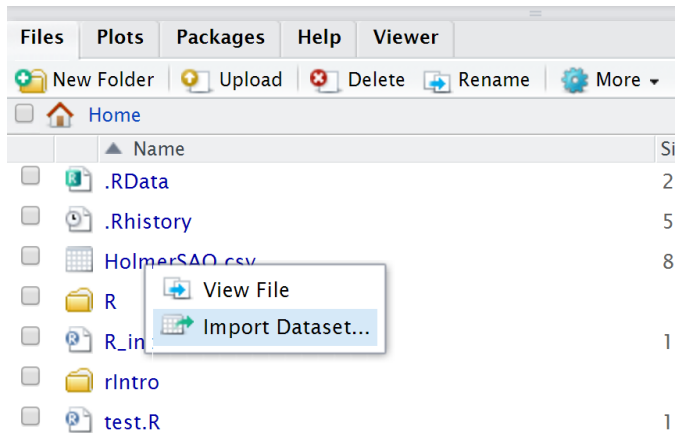


4- File > Import Dataset > From CSV...

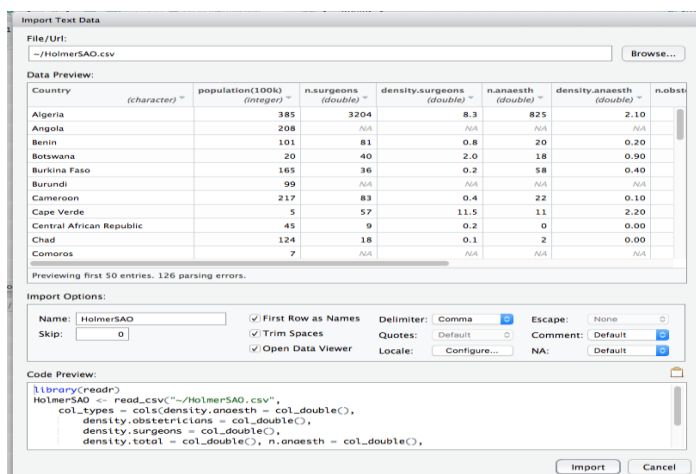




5-alternative. Or we can click on the file and Import Dataset.





























6- We can choose options like the delimiter in our file, assigned comment character, columns' data type, etc.



### Appendix 3: pch

As you see, pch = 19 that we used above is the filled circle.

0: 	10: 	20: 	A: <b>A</b>
1: 	11: 	21: 	a: <b>a</b>
2: 	12: 	22: 	B: <b>B</b>
3: 	13: 	23: 	b: <b>b</b>
4: 	14: 	24: 	S: <b>S</b>
5: 	15: 	25: 	`: `
6: 	16: 	@: <b>@</b>	.: .
7: 	17: 	+: <b>+</b>	,: ,
8: 	18: 	?: <b>%</b>	?: <b>?</b>
9: 	19: 	#: <b>#</b>	*: <b>*</b>

## Appendix 4: Swirl Package for Learning R within R

As mentioned, our focus is the web-based RStudio mostly to visualize and analyze data, but for students who want to learn more about R there is a useful interactive package called Swirl. This is a great package, but absolutely optional. You will be provided all about R you need to know in lecture and sections.

But if you still feel enthusiastic about R or to continue learning R after our summer journey, the following will have you up and running:

```
> library(swirl)
> swirl()
```