R Reference Guide

This R guide was put together to provide a quick reference guide to those who plan to use R in STAT 3011/3021. We focus here just on the types of things you will need to do in this course. There is much more you can do with R, however, and we hope that you'll feel comfortable exploring and trying new things once you get the basic ideas down (and that this exposure to R will provide you with a good foundation to study R in more detail in other courses).

As you are going through this guide, we strongly encourage you to walk through the steps we have outlined and practice using R (and ask questions!). We hope this will be a useful reference for you to refer back to many times this semester, and we also hope your feedback will help us to make this an even better reference for future students.

Also, please keep in mind that this guide is mostly meant to show you how to obtain certain kinds of output. You'll learn throughout the semester, by working through other activities and utilizing other resources, how to interpret this output.

Downloading R

Step 1: Go to http://www.r-project.org

Step 2: On the left hand side, click the link under the **Download, Packages** section that says **CRAN**

Step 3: Pick 0-Cloud https://cloud.r-project.org/

Step 4: Click on the system you have (Linux, Mac OS X, or Windows)

Step 5: For Mac, click on the Latest release (R-3.5.1.pkg as of

December 2018)

For Windows, click **base** and then **Download R for Window** (R 3.5.1 as of December 2018).

Step 6: Follow the directions for installing R onto your computer, and then you will be ready to go.

Downloading R Studio

Step 1: Go to https://www.rstudio.com/products/rstudio/download/

Step 2: On the left hand side, click DOWNLOAD button under RStudio Desktop Open Source License FREE.

Step 3: Pick a correct installer for your operating system (Windows, Mac, etc.)

Step 4: Follow the directions for installing RStudio onto your computer, and then you will be ready to go.

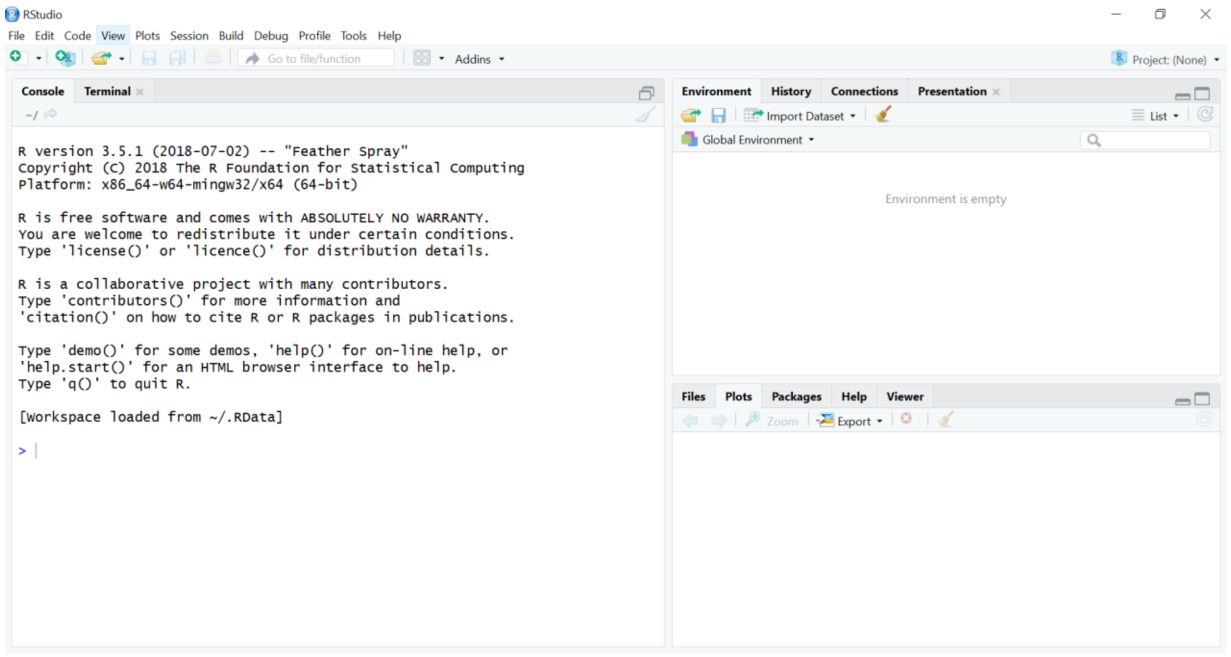
Getting Started

When you first open R Studio, like below you will see a screen with three windows; *Console* (left window), *Environment* (top right window), *Files/Plot* (bottom right window).

Console is where any work is processed and shows outputs. If there is any errors in your R commands, the error message will appear in the console.

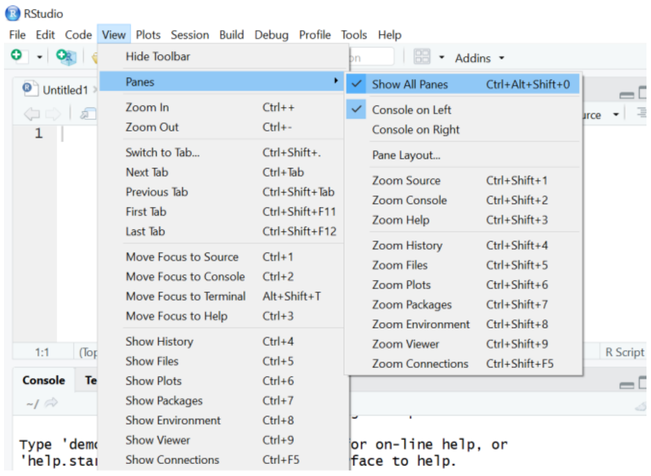
Environment shows datasets and objects created. You can click on '***History***' tab to view previous R commands.

Files show any files in your working directory by default. '***Plots***' tab shows any previous plots.



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 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.

[Workspace loaded from ~/.RData]

> 
```

If you do not see three windows, you can go do [View] - [Panes] – [View all Panes] (see below)



R as a calculator

Retype each line of the following R commands in your console and hit [Enter] and observe what R produces.

```
9+7
sqrt(16)
abs(-3)
3^2
3**2
3%2
3e1
3e-1
```

Questions:

What do you think '`sqrt(16)`' calculates?

What about '`abs(-3)`'?

Did you see any error message when you typed 3%2? Fix the command so it calculates 3 divided by 2.

What do you think '3e1' mean? What about '3e-1'?

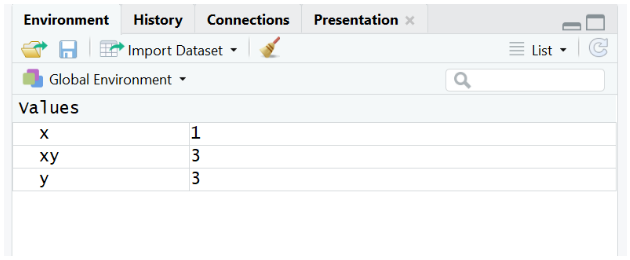
Note: Never enter > in console. This symbol indicates the program is ready for a new line of code and is called a *command prompt*.

Creating simple objects

You can use either ‘`<-`’ or ‘`=`’ to create objects.
Type each of the following line in your Console and heat [Enter]. Before you type the next command, check ‘Environment’ window if the object appears.

```
x<-1
y=3
xy<-x*y
```

Two commands above create two objects ‘`x`’ and ‘`y`’ and ‘`x`’ is defined as 1 and ‘`y`’ is defined as 3. ‘`xy`’ is defined as `x` multiplied by `y`. You can check ‘Environment’ Window (top right) to see all three objects (see below).



```
Values
  x        1
  xy       3
  y        3
```

Printing objects & adding comments

Try each line of commands below:

```
print(x)
x
y
y<-“a”
#now y is “a”. Check your Environment window.
y
typeof(y)
#Next X is capital
X
```

Note that both ‘`print(x)`’ and ‘`x`’ in line 1 and 2 return the stored value of `x` which is 1. Similarly ‘y’ returns the stored value of `y` which is `3.`
When you run ‘ `y=“a”` ’, R console **does not** print the value of `y`.
You can still see that the environment window has the updated the value of `y.`

‘#’ sign is used to comment. R ignores everything that appears after ‘#’. You can use the hash symbol at the beginning of a line or somewhere in the middle.

‘`typeof`’ is a built-in R function that returns the type of an argument.

R is case-sensitive; `x` and `X` are different objects. As you have not defined `X` yet, '`X`' will produce the following error message Error:

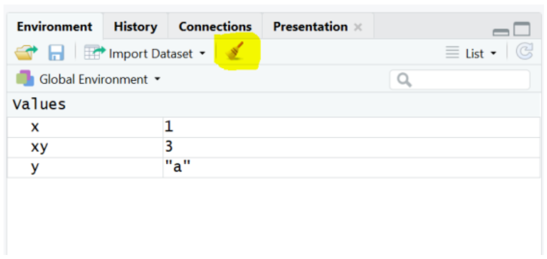
```
Error: object 'X' not found
```

Removing objects

To remove object(s), use '`rm`' or '`remove`' command:

```
rm(x) #removes x only
rm(x, y) #removes x and y
```

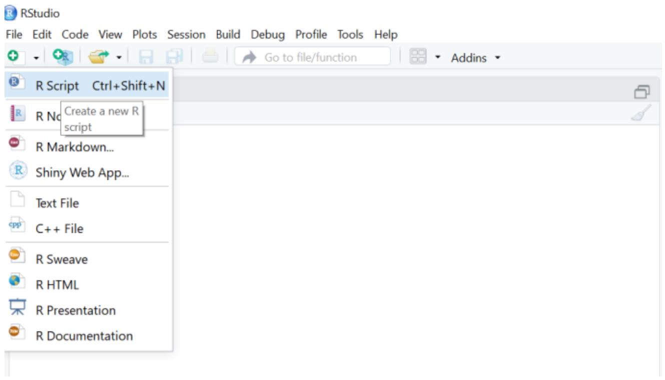
Verify that `x` and `y` are moved in your Environment window. You may click in Environment window to remove all stored objects and any dataset (see highlighted area below).

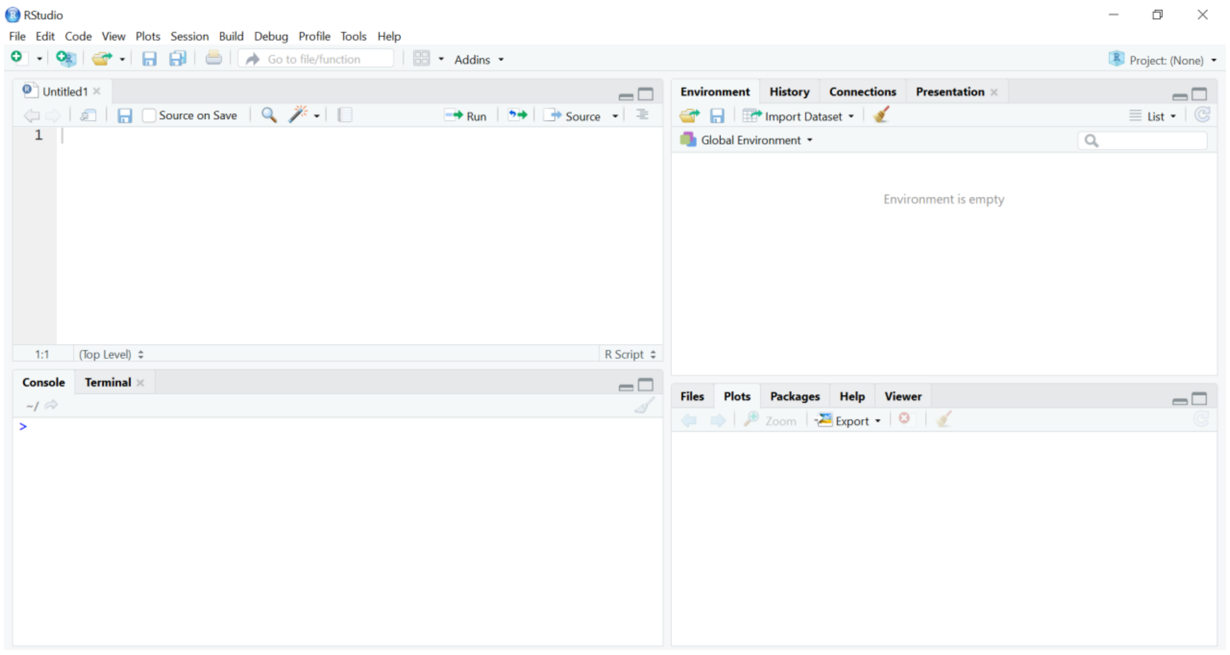


```
Values
  x        1
  xy       3
  y        "a"
```

Using R script

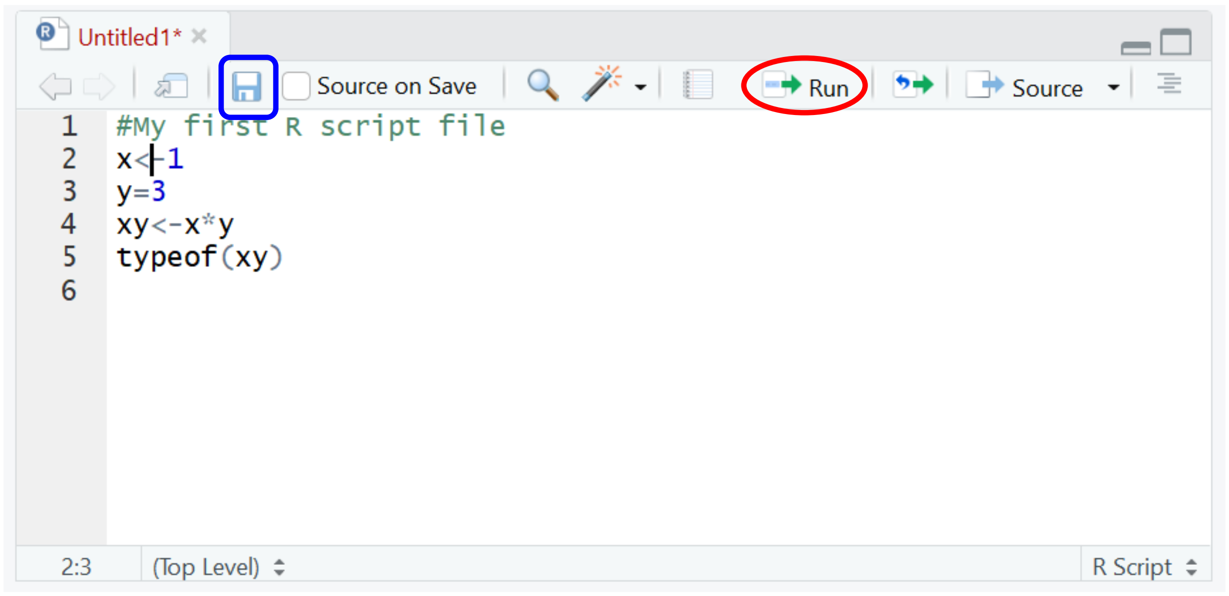
So far you've typed R commands in Console. Generally it is recommended to type R commands in R Script. R Script allows you to save your work and open the script file later and continue working on your project. You can reproduce outputs without typing commands again. To open R Script, click on the top left corner and click [R Script] (see below).





Environment is empty

You can write commands as you did in console. It is always good to explain what you are going to do as a comment. Instead of hitting [Enter], you can click 'Run' button (Red circle below). Make sure your cursor is at the command you want to run. You will see output in Console or Environment window. To run multiple commands, you can highlight all commands you want to run and click 'Run'. You can also run commands in Script by hitting [Ctrl]+[Enter] in Window PC or [Command]+[Enter] in Mac.



```
#My first R script file
x<-1
y=3
xy<-x*y
typeof(xy)

```

To save your script, click Save button (Blue rectangle above) or hit [Ctrl]+S. The next time you use R, you can simply open that script file (after you have opened R) and then add to it or run certain commands again.

Creating a vector

When you want to store more than one number/character to an object, you can create a vector. You can think of a vector as a row or column of numbers or text. The list of numbers {1,2,3,4,5}, for example, could be a vector. Unlike most other programming languages, R allows you to apply functions to the whole vector in a single operation (without a loop).

First assign the values of 1,2,3,4,5 to a vector called x:

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

Whenever you want to create a vector, you need '`c( )`'. The letter '`c`' stands for '*combine*'.

Type `x` in Console and hit [Enter]. You will see it returns 1,2,3,4,5.

Next, add the value 2 to each element in the vector x:

```
x+2
```

Create another vector y containing 6,7,8, 9, 10 and add vector x and y:

```
y<-6:10
```

```
x+y
```

Note that in `6:10` Colon operator is used instead of c(6,7,8,9, 10). `6:10` generates a sequence from 6 to 10.

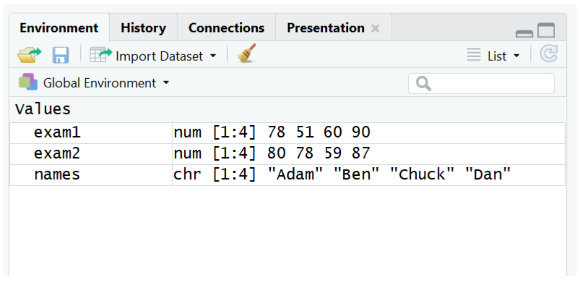
Creating different types of objects (entering data set manually)

Suppose you
have four students' names and their Exam 1 and Exam 2 scores.

```
names=c("Adam", "Ben", "Chuck", "Dan")
exam1=c(78, 51, 60, 90)
exam2<-c(80, 78, 59, 87)
```

Whenever you want to create a vector, you need '`c( )`'. The letter '`c`' stands for '*combine*'.

Check your Environment window to see if three objects are created properly (see below).



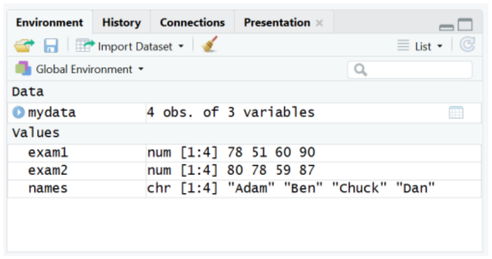
```
Values
  exam1       num [1:4] 78 51 60 90
  exam2       num [1:4] 80 78 59 87
  names       chr [1:4] "Adam" "Ben" "Chuck" "Dan"
```

'`num[1:4]`' indicates that each vector `exam1` and `exam2` contains 4 numbers. Similarly '`chr[1:4]`' shows that vector `names` contains 4 characters.

We can use a R built-in command '`data.frame()`' to make a dataset containing the three vectors.

```
mydata<-data.frame(names, exam1, exam2)
```

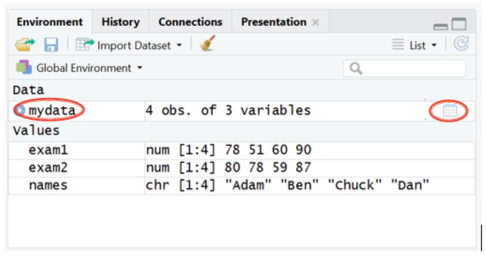
Check the Environment window and you will see a new data set '`mydata`'.



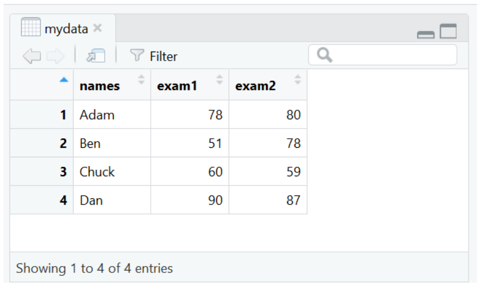
```
Data
 mydata       4 obs. of 3 variables
Values
  exam1       num [1:4] 78 51 60 90
  exam2       num [1:4] 80 78 59 87
  names       chr [1:4] "Adam" "Ben" "Chuck" "Dan"
```

`4 obs. of 3 variables` means that there are 4 observations (4 students) and 3 vectors (`names, exam1, exam2`) in 'mydata'

By clicking the name of the data set (see red circles below) or running R command '`View(mydata)`', you can see the entire data.



```
Data
 mydata       4 obs. of 3 variables
Values
  exam1       num [1:4] 78 51 60 90
  exam2       num [1:4] 80 78 59 87
  names       chr [1:4] "Adam" "Ben" "Chuck" "Dan"
```



	names	exam1	exam2
1	Adam	78	80
2	Ben	51	78
3	Chuck	60	59
4	Dan	90	87

Showing 1 to 4 of 4 entries

Tips and Tricks #1: Once an object has been created, it is always **good practice** to look what is contained in that object.

You can ALWAYS directly type the data into R, but working with an external file (like an Excel file) is very common (and you may see why when you find out how tedious it can be to enter a large data set with multiple variables directly into R). We thus encourage R users to get used to importing a data set into R.

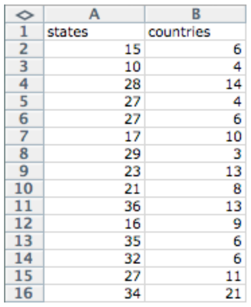
Data in Excel, imported into R

Suppose we asked all students in a class (where there are N = 15 students): "How many states and countries have you visited in your lifetime?" The following data was obtained from the class:

Number of states visited: 15 10 28 27 27 17 29 23 21 36 16 35 32 27 34

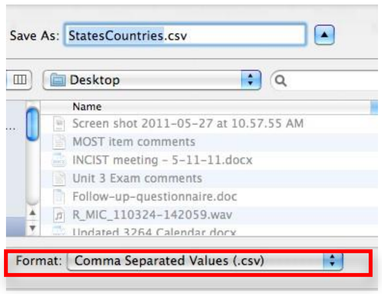
Number of countries visited: 6 4 14 4 6 10 3 13 8 13 9 6 6 11 21

Now there are two variables that we collected from the students: number of states and number of countries per student. Open Excel and type the variable names into the first row of the spreadsheet: `states` in row 1 column A, and `countries` in row 1 column B. Then enter the data, with each row representing one student in this example.



	A	B
1	states	countries
2	15	6
3	10	4
4	28	14
5	27	4
6	27	6
7	17	10
8	29	3
9	23	13
10	21	8
11	36	13
12	16	9
13	35	6
14	32	6
15	27	11
16	34	21

Once the data have been entered, save the Excel document as a .csv file: Go up to `File > Save As…`, then change the Format to Comma Separated Values (.csv) and name the file something meaningful. Note we show you below how to do this on the Mac; the ideas are the same with a PC, but with a PC, you might just see the letters CSV (or the words comma deliminated).



Save As: StatesCountries.csv

Desktop

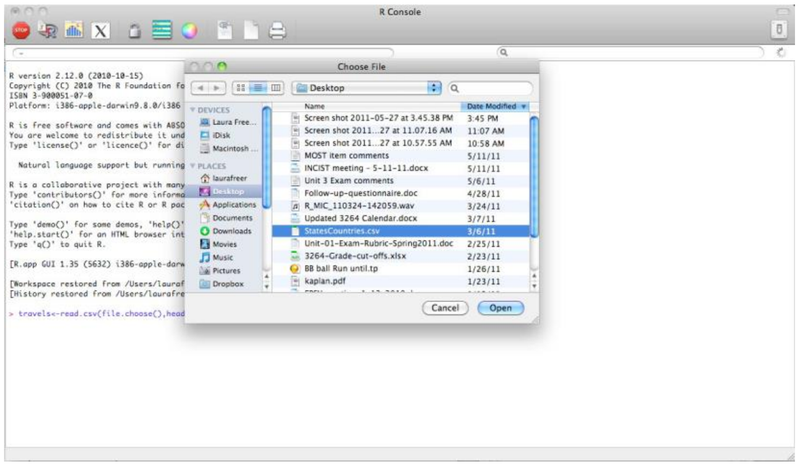
Name
Screen shot 2011-05-27 at 10.57.55 AM
MOST item comments
INCIST meeting - 5-11-11.docx
Unit 3 Exam comments
Follow-up-questionnaire.doc
R_MIC_110324-142059.wav
Updated 3264 Calendar.docx

Format: Comma Separated Values (.csv)

To import the file into R, type the following command (just as it is shown below) and run

```
travels<-read.csv(file.choose())
```

A window will pop up asking you to find the file that you wish to import.



```
R version 2.12.0 (2010-10-15)
Copyright (C) 2010 The R Foundation fo
ISBN 3-900051-07-0
Platform: i386-apple-darwin9.8.0/i386

R is free software and comes with ABSO
You are welcome to redistribute it und
Type 'license()' or 'licence()' for di

  Natural language support but running

R is a collaborative project with many
Type 'contributors()' for more informa
'citation()' on how to cite R or R pac

Type 'demo()' for some demos, 'help()'
'help.start()' for an HTML browser int
Type 'q()' to quit R.

[R.app GUI 1.35 (5632) i386-apple-darw

[Workspace restored from /Users/lauraf
[History restored from /Users/laurafre

> travels<-read.csv(file.choose(),head
```

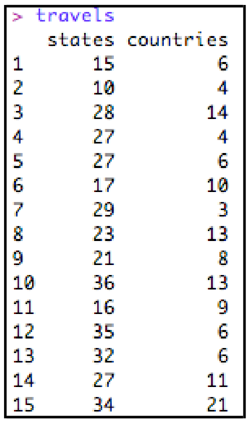
Name	Date Modified
Screen shot 2011-05-27 at 3.45.38 PM	3:45 PM
Screen shot 2011...27 at 11.07.16 AM	11:07 AM
Screen shot 2011...27 at 10.57.55 AM	10:58 AM
MOST item comments	5/11/11
INCIST meeting - 5-11-11.docx	5/11/11
Unit 3 Exam comments	5/6/11
Follow-up-questionnaire.doc	4/28/11
R_MIC_110324-142059.wav	3/24/11
Updated 3264 Calendar.docx	3/7/11
StatesCountries.csv	3/6/11
Unit-01-Exam-Rubric-Spring2011.doc	2/25/11
3264-Grade-cut-offs.xlsx	2/23/11
BB ball Run until.tp	1/26/11
kaplan.pdf	1/23/11

Find the file on your computer, click on it and then hit the OK button (or double-click the file name). **You know the file has been successfully imported if the name of the data set 'travels' appears in Environment window.**

The function `read.csv()` reads .csv files into R. We used `read.csv()` in the above example and stored the data under the object name of `travels`.

The argument for `read.csv()` tells R where the .csv file is located, on your computer or on the web. If the file is on your computer, the easiest way to find the file is through the function `file.choose()`. This is the function that opens up the window in the above example.

Back in the R console, type in the object name (`travels`) to look at the data that was just imported.



```
> travels
   states countries
1      15         6
2      10         4
3      28        14
4      27         4
5      27         6
6      17        10
7      29         3
8      23        13
9      21         8
10     36        13
11     16         9
12     35         6
13     32         6
14     27        11
15     34        21
```

The object `travels` is a *data frame*. It contains two vectors, `states` and `countries`. Less informally, you can think of a data frame as a table that contains rows as observations and columns as variables.

Tips and Tricks #2: When you name variables, try to make them short (usually one word), simple, and interpretable. In the above example, here are two different versions of the variable names:

Short version (good): states and countries

Long version (too long): Number_of_states and Number_of_countries

However, if you insist on including the word Number in the variable name, then here are three shorter examples:

Period separator: num.states and num.countries

Underscore separator: num_states and num_countries

Capitalization separator: NumStates and NumCountries

Lastly, never name objects a name of a function (e.g., mean, sd, median, c, etc.). If you aren't sure if it's a function name, Google it (e.g., mean R).

Entering data from a URL into R

Data can be imported into R using the 'read.table()', 'read.csv()', or 'read.delim()' depending on how the data is structured. For example, if you want to import Australian crime data set into R, you can use a name of the data (say "au_crime") set and type the following :

```
au_crime<-read.table("http://www.stat.umn.edu/~wuxxx725/data/crime.txt",
header=T)
```

The second argument 'header=T' tells R that the file contains column labels (or headers).

Graphing Data

One of the major strengths of R is its graphing capabilities. The main focus of graphical displays in this class will be for quantitative data. In particular, we'll look at boxplots and histograms.

Creating Boxplots: Single and Side-by-Side Boxplots

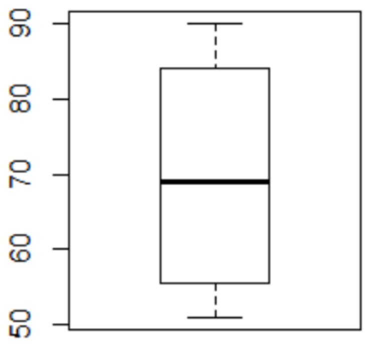
In the states/countries example above, we'd ideally like side-by-side boxplots. We gathered information about two variables, and both variables are measured in the same units. Thus, it makes sense to create side-by-side graphs so we can easily compare the two variables in terms of features like shape, center, and variability.

Before the side-by-side syntax is introduced, we will first create a boxplot with a single variable.

If your data were directly entered into R, as you did with exam 1 and exam 2 data, then you can use following commands :

```
boxplot(exam1)
```

Then you will see a boxplot in 'Plots' lab.

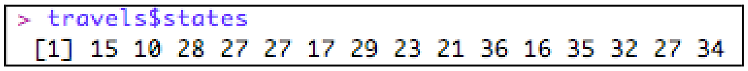


The name of the function for creating boxplots is called `boxplot()` and there is only one argument needed to create a single variable boxplot – the name of the single variable vector (`exam1`). The syntax `boxplot(exam1)` will take the 4 numbers from the variable `exam1` and create a boxplot from them.

If your data was imported into R, as you did in `travel` data, then the variable names cannot be directly accessed, that is, you cannot just type `states` or `countries` and get the respective data. The variables must be referenced according to the object that contains the variables. If we only want to look at the `states` variable/vector, then type the (object name)$(variable name), as you see here:

```
travels$states
```

If you simply run `travels$states`, you will then see just the column of data within travels that contains the `states` variable in the console, as shown below.

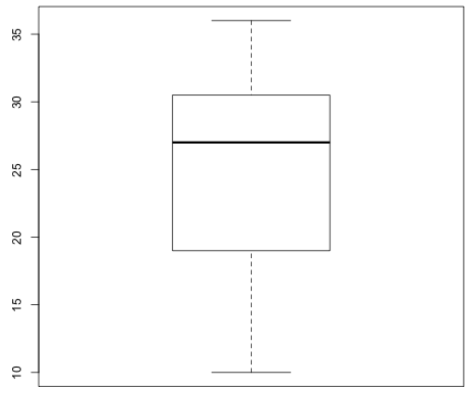


```
> travels$states
 [1] 15 10 28 27 27 17 29 23 21 36 16 35 32 27 34
```

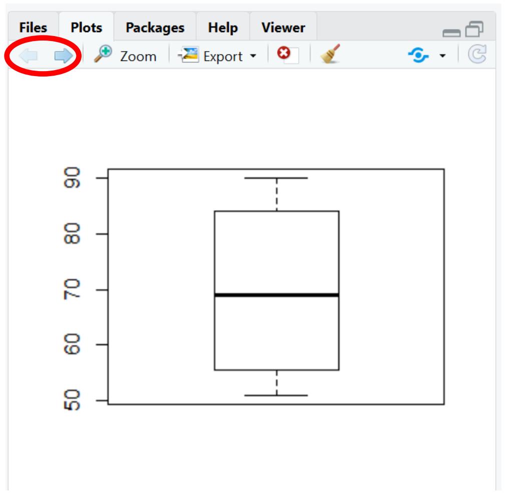
Then, to plot `states` using the `boxplot()` function, you'd simply type the following at the command prompt and then hit Enter/Return:

```
boxplot(travels$states)
```

As you see, we get the boxplot in the Plots tab.



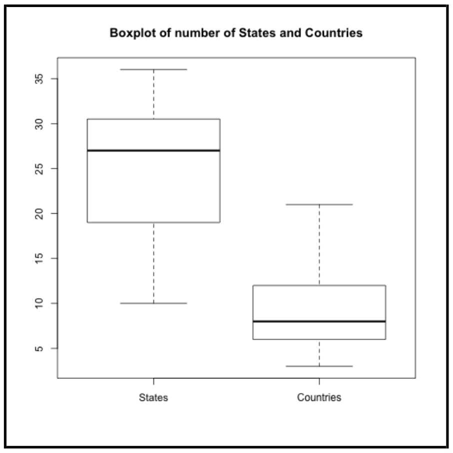
You can switch between two plots by clicking an arrow in Plots window (red circle below).



When you have two variables that are measured using the same units (such as `states` and `countries`), you may want to compare the two variables side-by-side with side-by- side boxplots (i.e., a plot of states right next to a plot of countries). To do this in R, run the following syntax (note the syntax here is long, and you will want to type it just as you see it below, together on one line):

```
boxplot(travels$states,travels$countries,names=c("States","Countr
ies"),main="Boxplot of number of States and Countries")
```

Doing this will give you the following boxplot.



The `boxplot()` function was used again. Then the minimal arguments needed for creating side-by-side boxplots is the first line in the above syntax, `boxplot(travels$states, travels$countries)`.

The next two lines are optional arguments. The `names=` argument puts labels under each plotted boxplot. Since we have more than one label, the `c()` function needs to be used in order to put multiple labels on the graph. The `main=` argument will put a title on the plot window. Remember to use quotes around labels or titles.

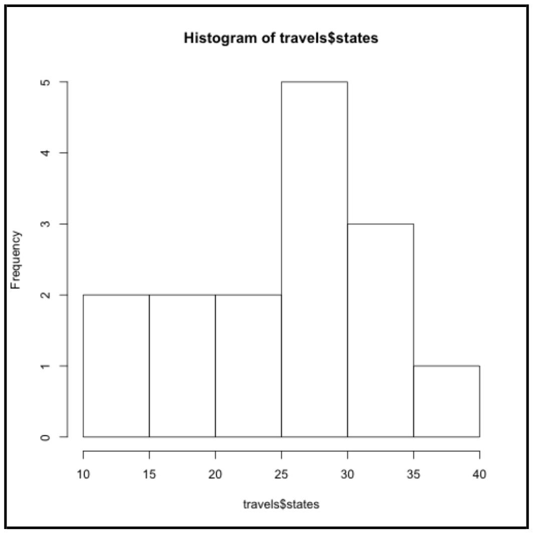
There are other graphical options for creating boxplots. Type `?boxplot` to access the boxplot R help page for more details.

Creating Histograms

Instead of boxplots, we could have created histograms for each variable individually. For example, to create a histogram of states, we run the following syntax:

```
hist(travels$states)
```

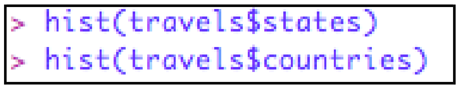
The following histogram will be displayed once you execute the above command.



If we want to see the histograms in the same plot window, we can adjust the plot window options in R. The following syntax will change the plot window options from 1 plot in the plot window to two plots in the plot window, 1 row and 2 columns:

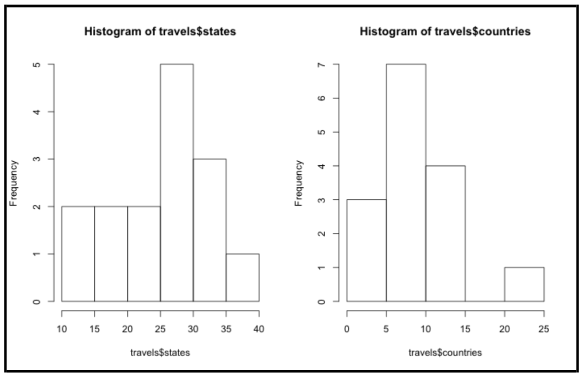
```
par(mfrow=c(1,2))
```

Then we can call the `hist()` function twice, one for each variable, to get two plots in the plot window, as shown below (note that you would type in each command on a new line).

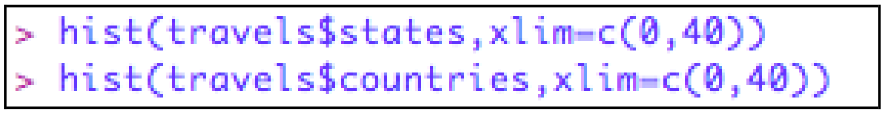


```
> hist(travels$states)
> hist(travels$countries)
```

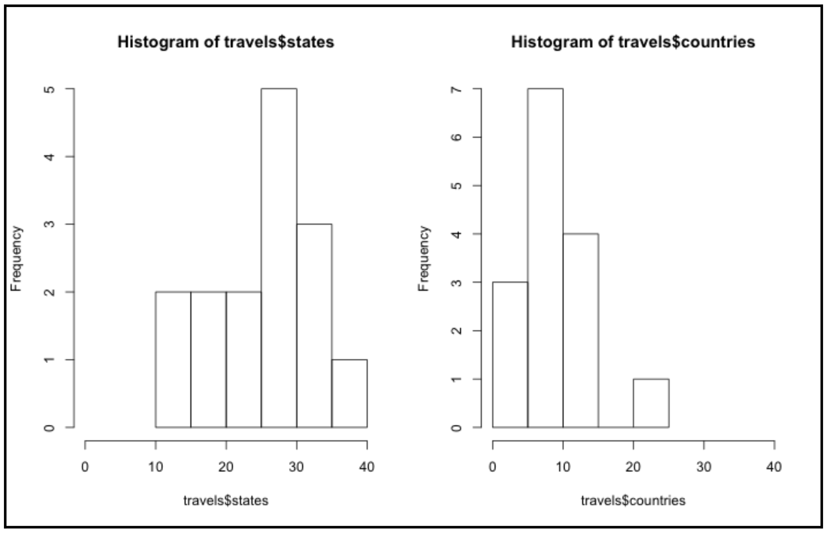
This will give us the following output, and this will allow us to do side-by-side comparisons of our variables.



If you are looking at two variables that were measured in the same units, you might want to specify that the x-axes in the histograms are similar. In the below code, we specify the limits for the x-axis using the `xlim=c(lower limit, upper limit)` argument.



```
> hist(travels$states,xlim=c(0,40))
> hist(travels$countries,xlim=c(0,40))
```



After you are finished with creating multiple plots in the plot window, make sure to change the plot options back to the default options, by typing in the following command and then hitting Enter/Return:

```
par(mfrow=c(1,1))
```

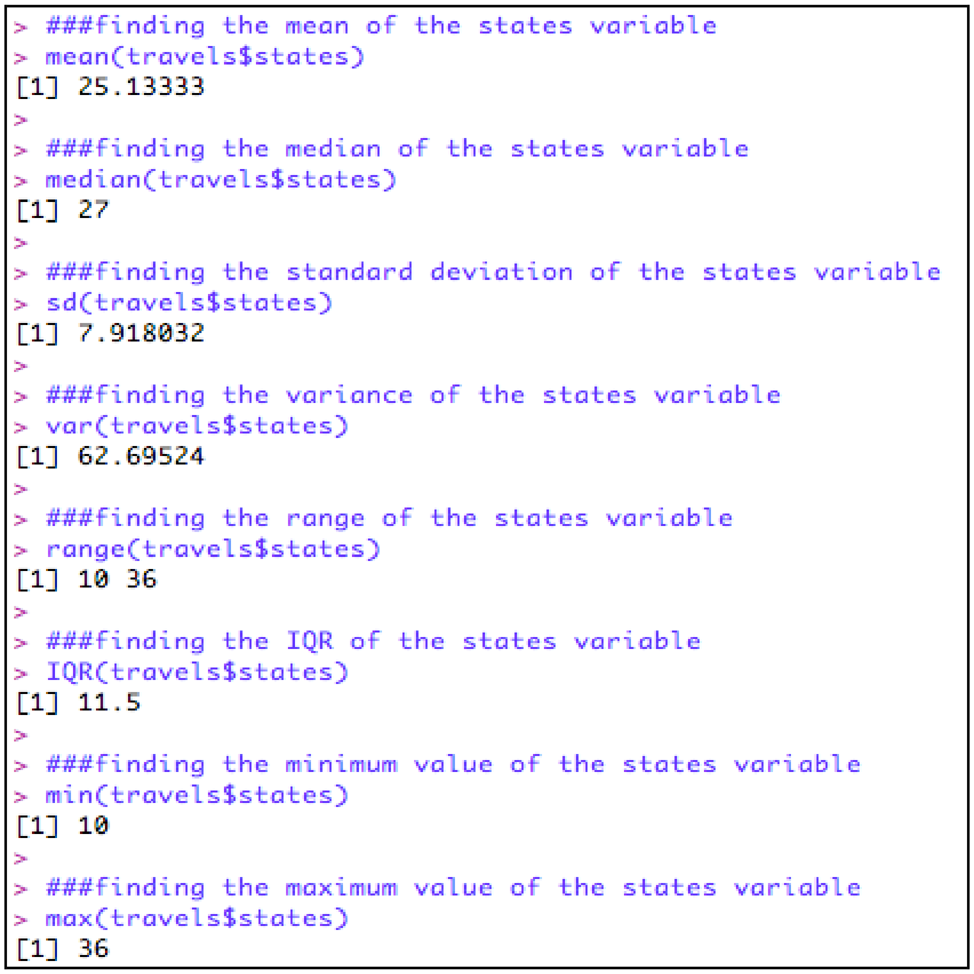
As you create graphs (and other output), you should always keep in mind that there are more options/arguments for each function. You can access what options are available by looking at the R help page for that function. Type `?(function name)`. For example, `?hist`.

Copying and Pasting Output from R into Word

It is easy to copy R graphs and put them in many applications (e.g., Microsoft Word document). If you select the R graphics window by clicking on "Export"-[Copy to clipboard…], you can select the size you want and click "Copy Plot". Then **Paste** the graph into the application document.

Summarizing Data with Numbers

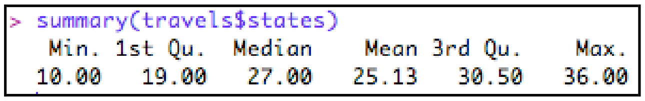
In addition to graphing data, we will often want to summarize our data numerically by computing various measures of center (like the mean and median) and variability (like the standard deviation, range, and interquartile range). This is done by typing each summary function out on different lines of code. Below are some examples of what this would look like in R.



```
> ###finding the mean of the states variable
> mean(travels$states)
[1] 25.13333
>
> ###finding the median of the states variable
> median(travels$states)
[1] 27
>
> ###finding the standard deviation of the states variable
> sd(travels$states)
[1] 7.918032
>
> ###finding the variance of the states variable
> var(travels$states)
[1] 62.69524
>
> ###finding the range of the states variable
> range(travels$states)
[1] 10 36
>
> ###finding the IQR of the states variable
> IQR(travels$states)
[1] 11.5
>
> ###finding the minimum value of the states variable
> min(travels$states)
[1] 10
>
> ###finding the maximum value of the states variable
> max(travels$states)
[1] 36
```

Tips and Tricks #3: You may want to add comments within your code or script file so that you can recall at a later time what was executed. To do this, type # (pound sign) and R will ignore everything after #. You only need one # in order to make a comment.

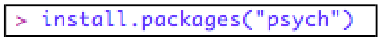
There is another summary function that gives you the 5-number summary (minimum, 1st quartile, median, 3rd quartile, maximum), plus the mean. That command is shown below.



```
> summary(travels$states)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  10.00   19.00   27.00   25.13   30.50   36.00 
```

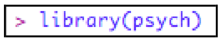
The final function option for obtaining summary measures, and the best one of them all, is called the `describe()` function. But first, we must learn about packages in R and install the `{psych}` package.

Packages are collections of R functions and data. R comes with a standard set of packages that are ready to use right after installation of the software. However, there are other packages available that could be useful to the R user. The package that is useful to us in this particular instance is the `{psych}` package. To obtain the functions and data within this package, we need to do a one-time only installation of the package onto our computer. Use the function `install.packages("package name")` to download and install packages.



```
> install.packages("psych")
```

Then select a CRAN Mirror of which you want to download the package from (e.g., Iowa State, or USA(IA)). Finally, type the following to access the functions and data that are contained in the `{psych}` package:



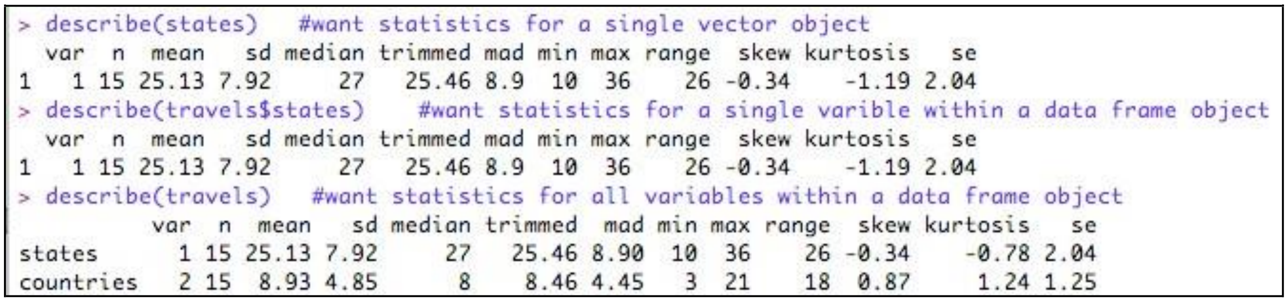
```
> library(psych)
```

This above line of code must be rerun with every new R session if you want to use the functions and data within the package. That is, if you exit R and then come back to it and want to use the `describe()` function, then you must first run `library(psych)`.

The `describe()` function provides the following summary statistics for either vector objects or data frame objects:

- `var`: variable id
- `n`: number of rows or units in the object
- `mean`: mean of the variable(s) in the object
- `sd`: standard deviation of the variable(s) in the object
- `median`: median of the variable(s) in the object
- `trimmed`: 10% trimmed mean of the variable(s) in the object
- `mad`: median absolute deviation of the variable(s) in the object
- `min`: minimum value of the variable(s) in the object
- `max`: maximum value of the variable(s) in the object
- `range`: maximum – minimum calculation of the variable(s) in the object
- `skew`: skew statistic of the variable(s) in the object
- `kurtosis`: kurtosis statistic of the variable(s) in the object
- `se`: standard error of the variable(s) in the object

Below, you'll see some examples of how we can use this function with our states and countries data file.



```
> describe(states)   #want statistics for a single vector object
  var  n  mean   sd median trimmed mad min max range  skew kurtosis   se
1   1 15 25.13 7.92     27   25.46 8.9  10  36    26 -0.34    -1.19 2.04
> describe(travels$states)    #want statistics for a single varible within a data frame object
  var  n  mean   sd median trimmed mad min max range  skew kurtosis   se
1   1 15 25.13 7.92     27   25.46 8.9  10  36    26 -0.34    -1.19 2.04
> describe(travels)   #want statistics for all variables within a data frame object
          var  n  mean   sd median trimmed  mad min max range  skew kurtosis   se
states      1 15 25.13 7.92     27   25.46 8.90  10  36    26 -0.34    -0.78 2.04
countries   2 15  8.93 4.85      8    8.46 4.45   3  21    18  0.87     1.24 1.25
```

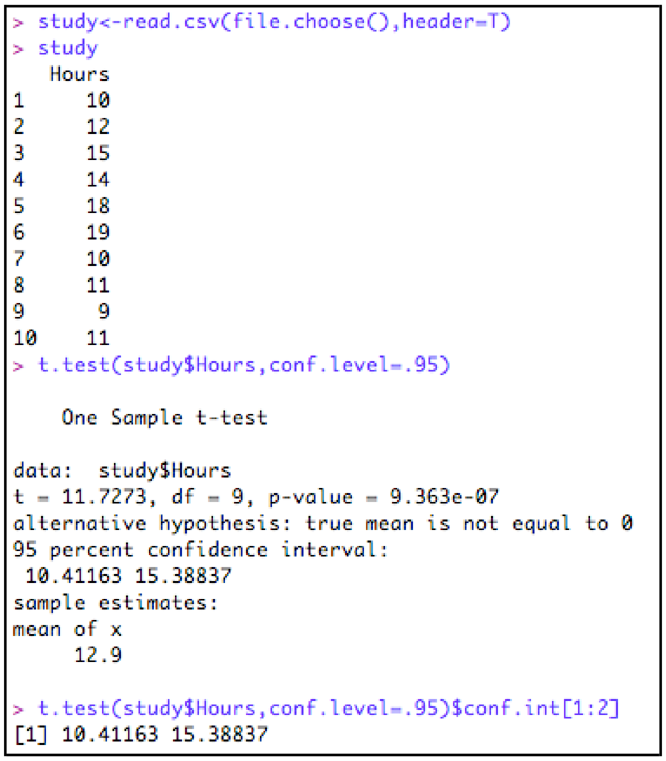
Creating a Confidence Interval for a Single Quantitative Variable

Suppose you collected data on the number of hours students indicate they have studied for an exam:

10 12 15 14 18 19 10 11 9 11

Recall that the data can be directly typed into R using the `c()` function, or it can be imported in if the data is in a .csv file.

The following example will import the above data from a .csv file into a data frame object called `study`. Then, a 95% confidence interval will be computed for the `Hours` variable.



```
> study<-read.csv(file.choose(),header=T)
> study
   Hours
1     10
2     12
3     15
4     14
5     18
6     19
7     10
8     11
9      9
10    11
> t.test(study$Hours,conf.level=.95)

	One Sample t-test

data:  study$Hours
t = 11.7273, df = 9, p-value = 9.363e-07
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 10.41163 15.38837
sample estimates:
mean of x
     12.9

> t.test(study$Hours,conf.level=.95)$conf.int[1:2]
[1] 10.41163 15.38837
```

The `t.test()` function provides a lot of useful information regarding the `Hours` variable. The function gives us the *t*-test information (*t*-statistic, df, *p*-value, hypothesized value), confidence interval, and sample mean statistic. The only argument that is required in the function is the variable we want to analyze (i.e, the `Hours` variable in the data set `study`: `study$Hours`).

By default, a 95% confidence level is computed, but you can change the level of confidence by typing in a new value. For example, if you would rather have a 90% confidence level, type `conf.level = .90` rather than `conf.level=.95`.

The last line of code, `t.test(study$Hours, conf.level=.95)$conf.int[1:2]`, specifies that we only want the confidence interval boundaries and not the rest of the *t*- test information. As you can see, the syntax outputs only the lower and upper limits of the confidence interval.

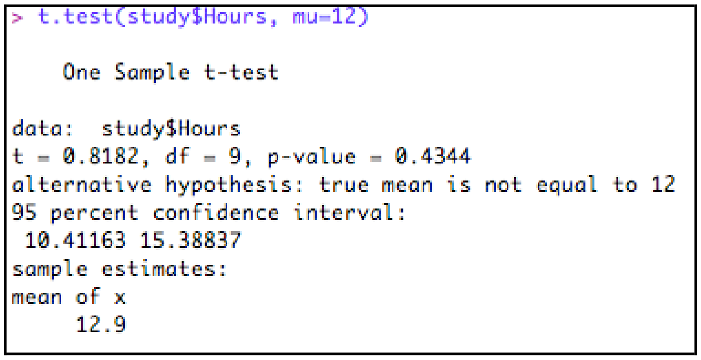
<u>Conducting a One-sample *t*-test</u>

Suppose there is a claim that students tend to study 12 hours for particular exam in question. You want to test this claim with your sample of data. Since you have just a single sample and you are using this sample to test a claim about a population parameter (the population mean), you will conduct a one-sample *t*-test, with the following hypotheses:

H_0: $\mu = 12$

H_a: $\mu \neq 12$

You will once again use the `t.test()` function. By default, the population parameter is set to not equal to 0 (i.e., parameter ≠ 0) in the alternative hypothesis (a two-sided or two-tailed test).



```
> t.test(study$Hours, mu=12)

	One Sample t-test

data:  study$Hours
t = 0.8182, df = 9, p-value = 0.4344
alternative hypothesis: true mean is not equal to 12
95 percent confidence interval:
 10.41163 15.38837
sample estimates:
mean of x 
     12.9 
```

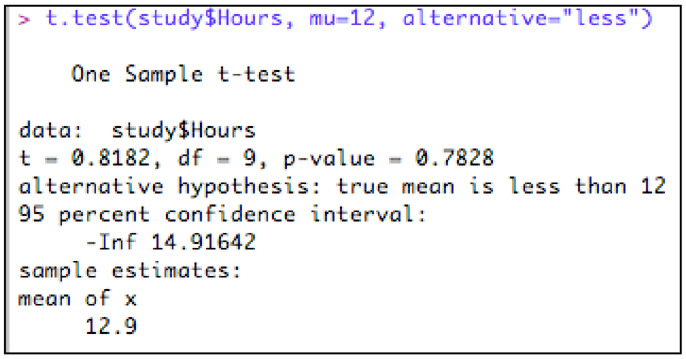
The above can be modified if we want to test different alternative hypotheses, other than the two-sided.

If we want to test the "less than" hypothesis,

H_0: $\mu = 12$

H_a: $\mu < 12$

type in the following:



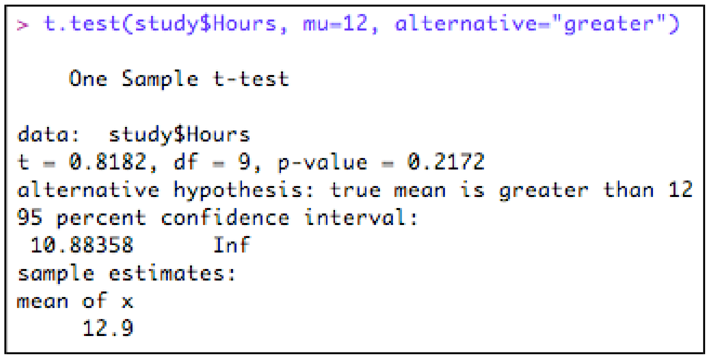
```
> t.test(study$Hours, mu=12, alternative="less")

	One Sample t-test

data:  study$Hours
t = 0.8182, df = 9, p-value = 0.7828
alternative hypothesis: true mean is less than 12
95 percent confidence interval:
     -Inf 14.91642
sample estimates:
mean of x
     12.9
```

If we want to test the “greater than” hypothesis,

H_0: $\mu = 12$

H_a: $\mu > 12$

type in the following:



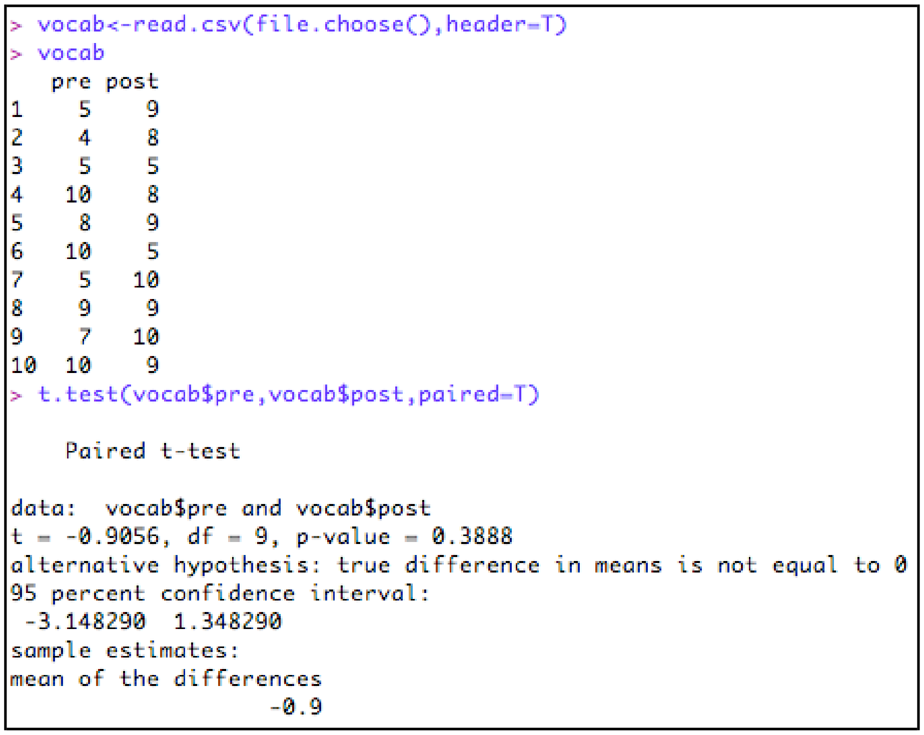
```
> t.test(study$Hours, mu=12, alternative="greater")

	One Sample t-test

data:  study$Hours
t = 0.8182, df = 9, p-value = 0.2172
alternative hypothesis: true mean is greater than 12
95 percent confidence interval:
 10.88358      Inf
sample estimates:
mean of x
     12.9
```

Conducting a Paired *t*-test

Ten children have taken a pre-test of their vocabulary knowledge and a post-test of their vocabulary knowledge. You would like to conduct a paired *t*-test to determine if there has been a significant change in knowledge from pre-test to post-test. Since each child was measured twice, the paired *t*-test is the appropriate technique to use here. We can conduct a *t*-test, and we can also construct a confidence interval from the mean difference.

H_0: $\mu_d = 0$

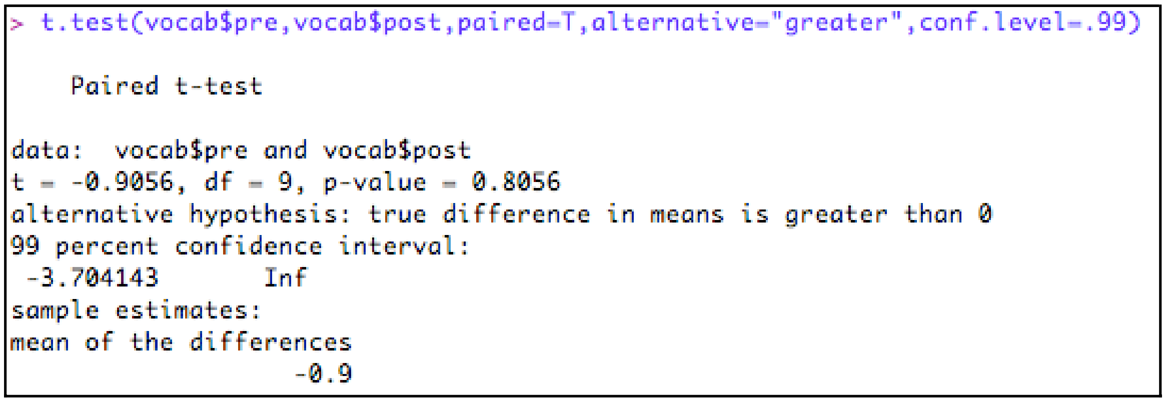
H_a: $\mu_d \neq 0$


```
> vocab<-read.csv(file.choose(),header=T)
> vocab
   pre post
1    5    9
2    4    8
3    5    5
4   10    8
5    8    9
6   10    5
7    5   10
8    9    9
9    7   10
10  10    9
> t.test(vocab$pre,vocab$post,paired=T)

	Paired t-test

data:  vocab$pre and vocab$post
t = -0.9056, df = 9, p-value = 0.3888
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.148290  1.348290
sample estimates:
mean of the differences 
                   -0.9 
```

Similar to finding confidence intervals and conducting a one-sample *t*-test in R, the `t.test()` function is used for conducting a paired t-test and finding a confidence interval for the paired data. In this situation, we provide the two variables that were collected for each child (`vocab$pre` and `vocab$post`) and also specify that we have paired data (`paired = T` argument). The function takes the first argument (`vocab$pre`) and subtracts the second argument from the first (`vocab$post`): `vocab$pre - vocab$post`. Finally, as seen previously with the `t.test()` function, the arguments of the function may be modified to perform the appropriate analysis (e.g., confidence level, direction of the alternative). See the example below that changes the direction of the alternative to greater than 0 and changes the confidence level to 99%.

H_0: $\mu_d = 0$

H_a: $\mu_d > 0$


```
> t.test(vocab$pre,vocab$post,paired=T,alternative="greater",conf.level=.99)

	Paired t-test

data:  vocab$pre and vocab$post
t = -0.9056, df = 9, p-value = 0.8056
alternative hypothesis: true difference in means is greater than 0
99 percent confidence interval:
 -3.704143       Inf
sample estimates:
mean of the differences 
                   -0.9 
```

Note that if you have paired data, you will need to find the differences for each pair and then graph those differences in order to determine if you are violating the necessary data condition of normality in the population. Suppose we want to determine what the differences are with our data object *vocab*.

You will need to specify the columns from the *vocab* object that you would like to perform arithmetic on (in order to compute the differences for each pair of observations). When doing arithmetic on the two vectors, it will take the difference of the "like" rows. So post[row 1] - pre[row 1] = 9 - 5 = 4, and post[row 2] - pre[row 2] = 8 - 4 = 4, etc. Arithmetic cannot be applied to two vectors that aren't of the same length.

To find all the differences, simply type the following command at the prompt:

```
vocab$post-vocab$pre
```

If you do this and hit the Enter/Return key, you'll see all the differences were computed.

```
[1] 4 4 0 -2 1 -5 5 0 3 -1
```

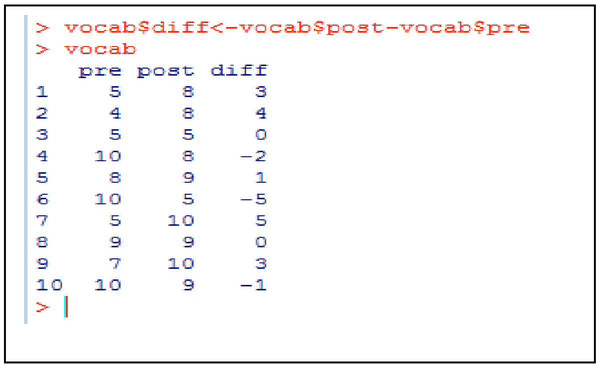
You can then graph the differences by simply typing something like the following:

```
hist(vocab$post-vocab$pre)
```

Alternatively, you can create a new column in *vocab*, by typing *vocab$(new column name)* <-, and then specify what you want in that column. In this case, I want a new column and I am calling it *diff*, and I want the difference between the post and the pre test to be in this column. I would first type in the following command at the command prompt:

```
vocab$diff<-vocab$post-vocab$pre
```

Now, when I look at my *vocab* object, by typing in vocab at the command prompt, I will see the following:

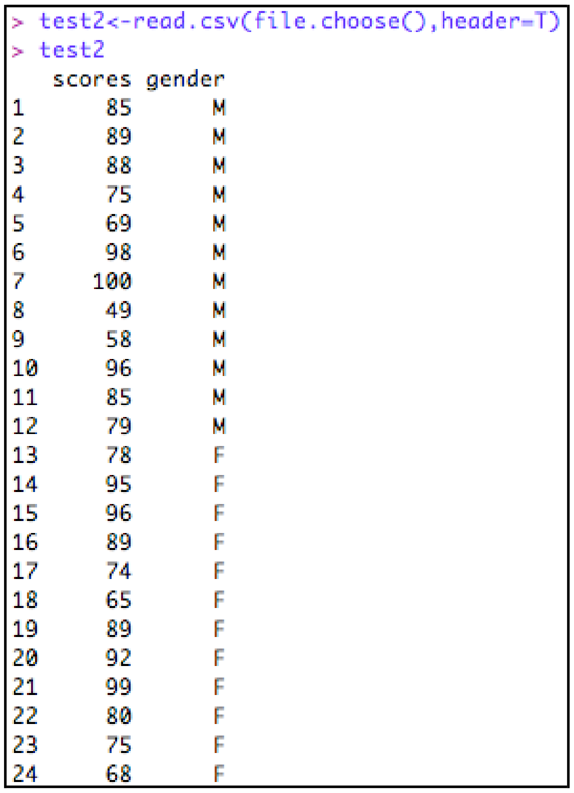


```
> vocab$diff<-vocab$post-vocab$pre
> vocab
   pre post diff
1    5    8    3
2    4    8    4
3    5    5    0
4   10    8   -2
5    8    9    1
6   10    5   -5
7    5   10    5
8    9    9    0
9    7   10    3
10  10    9   -1
> 
```

Conducting a Two-sample *t*-test

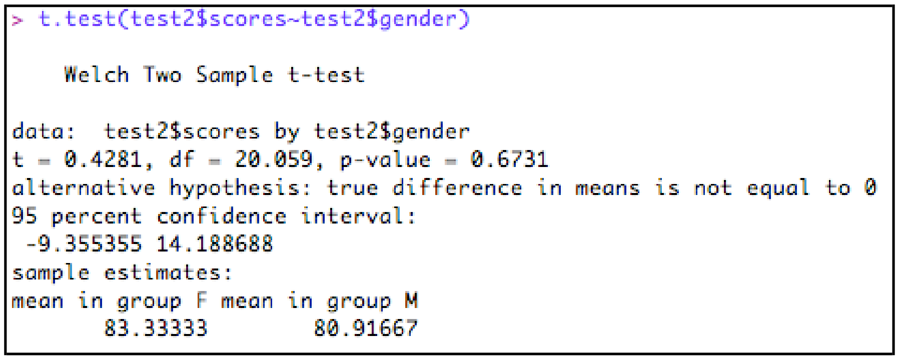
Suppose you have obtained a random sample of male students and a random sample of female students and you would like to compare these two groups on scores on a particular test. In this case, we have two independent samples – there is no reason for us to believe each male in one sample would be paired or matched with one particular female in the other sample. A two-sample *t*-test is therefore in order.

There are different ways to enter two-sample data in order to conduct a two-sample *t*-test. We'll show you one way here because we want you to get used to entering data in a particular way when you are comparing independent groups. Here, you structure your data by putting all of the scores for both groups in one column, and then you have a second column that identifies the gender for that particular score. Note that below, we are using "M" and "F" to stand for males and females, respectively. You could also use numerical codes if you wanted (i.e., you could type in a 1 to stand for Males and a 2 for Females).



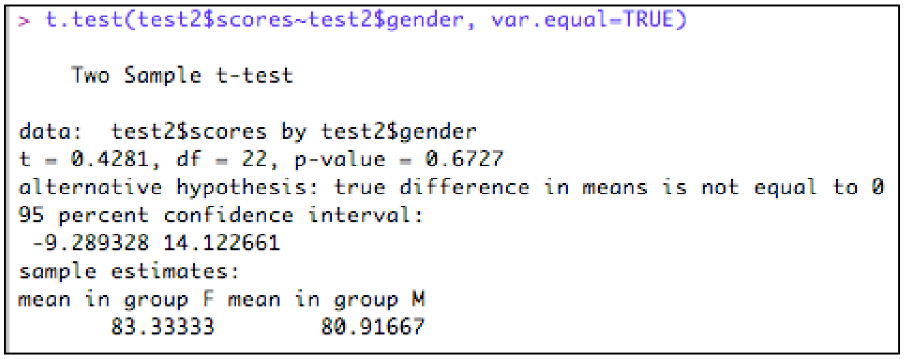
```
> test2<-read.csv(file.choose(),header=T)
> test2
   scores gender
1      85      M
2      89      M
3      88      M
4      75      M
5      69      M
6      98      M
7     100      M
8      49      M
9      58      M
10     96      M
11     85      M
12     79      M
13     78      F
14     95      F
15     96      F
16     89      F
17     74      F
18     65      F
19     89      F
20     92      F
21     99      F
22     80      F
23     75      F
24     68      F
```

If you enter the data as shown above, the appropriate syntax for conducting the two-sample t-test is:


```
> t.test(test2$scores~test2$gender)

	Welch Two Sample t-test

data:  test2$scores by test2$gender
t = 0.4281, df = 20.059, p-value = 0.6731
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -9.355355 14.188688
sample estimates:
mean in group F mean in group M 
       83.33333        80.91667 
```

As in the previous 3 sections, the arguments in the `t.test()` function may be changed to fit our analysis (e.g., confidence level, direction of alternative). By default, the two- sample *t*-test in R **assumes unequal variances**.

The assumption of equal population variances between the two groups is usually not valid. However, if you wish to assume equal variances, use the syntax below that modifies the `var.equal=` argument in the `t.test()` function.


```
> t.test(test2$scores~test2$gender, var.equal=TRUE)

	Two Sample t-test

data:  test2$scores by test2$gender
t = 0.4281, df = 22, p-value = 0.6727
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -9.289328 14.122661
sample estimates:
mean in group F mean in group M 
       83.33333        80.91667 
```

Separate Summary Statistics and Graphs for Each Sample

If you are using R for your project and you have two independent samples, or groups, you will need to spend some time describing the distributions of each sample separately (i.e., we want you to graph each sample by itself, compute summary statistics for each sample by itself, and use this information to describe the sample; you will then be able to compare and contrast the two samples based on what you see in the graphs and based on the different measures of center and variability you compute).

There are two ways you can do this in R.

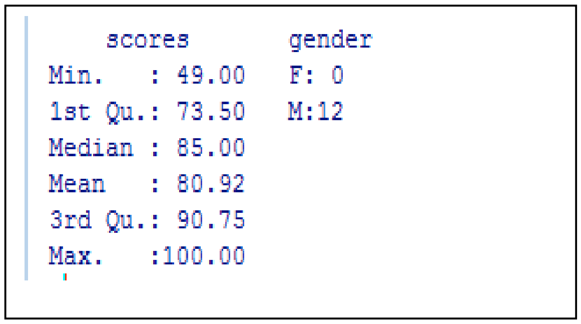
First, you can specify which rows in the data object file correspond to the gender of interest (i.e., the gender you want to graph or compute summary statistics for).

Let’s go back to the *test2* data object. We can specify which rows to look at by using the [rows, columns] after the data object name. Below, you will see some syntax that you would need to type in to do this for males.

```
summary(test2[test2$gender=="M",1])
```

With the above syntax, we are basically telling R that we want summary information on the *test2* object, but only looking at the rows where gender is Male (here we need == (or double "equal" signs) instead of just = because it's a logical statement, and we need quotes around the M because it's not a number). The 1 means that we only want output for the scores variable (i.e., we want to know the summary statistics for the scores for males) which is in column 1 in our spreadsheet.

If you type in the above command and hit Enter/Return, you'll get the following output:



```
    scores        gender
 Min.   : 49.00   F: 0  
 1st Qu.: 73.50   M:12  
 Median : 85.00         
 Mean   : 80.92         
 3rd Qu.: 90.75         
 Max.   :100.00         
```

Certainly, you can do the same for the female data; same syntax, just change M to F. You can also graph each group separately if you want to. For example, if you wanted a boxplot for males, you'd simply type the following command:

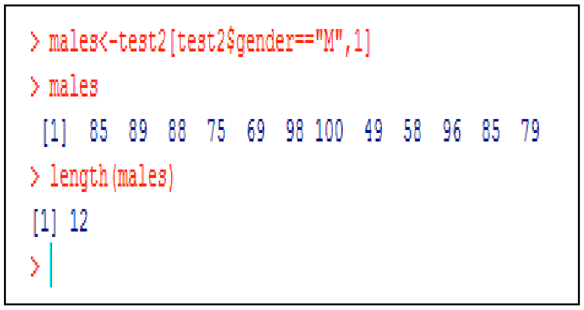
```
boxplot(test2[test2$gender=="M",1])
```

Earlier in this reference guide, we showed you how you can get even more summary statistics for your variables (other than just the five-number summary). Do you see now how you might get those values in this example, for the males in the data set?

Second, instead of doing what we just did, you could make two NEW separate data objects so you can simplify the typing. Certainly, typing *test2[test2$gender=="M",1]* can be used for any function and it will give you all of the information you need for the males in the study. However, if you want to use this syntax multiple times (for descriptive or summary statistics, histograms, etc.), it is probably easiest if you create a NEW data object that only contains the Male data information (and then do the same for the second group, the Females). To do this here, simply type the following command:

```
males<-test2[test2$gender=="M",1]
```

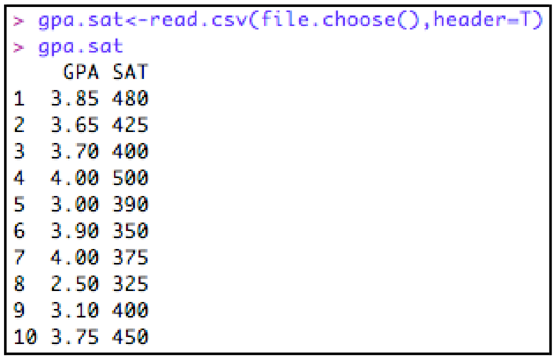
If you then type in *males* at the command prompt (and hit Enter/Return), you'll see just the scores (i.e., the values in column 1) for males. If you then type *length(males)* and hit Enter/Return, you'll see the number of cases we have for males (here it is 12).


```
> males<-test2[test2$gender=="M",1]
> males
 [1]  85  89  88  75  69  98 100  49  58  96  85  79
> length(males)
[1] 12
> 
```

Hopefully you are now getting the hang of things and you see how you could also do this for the females in the data set AND find summary statistics for each gender (and graph each gender).

<u>Correlation and Simple Linear Regression</u>

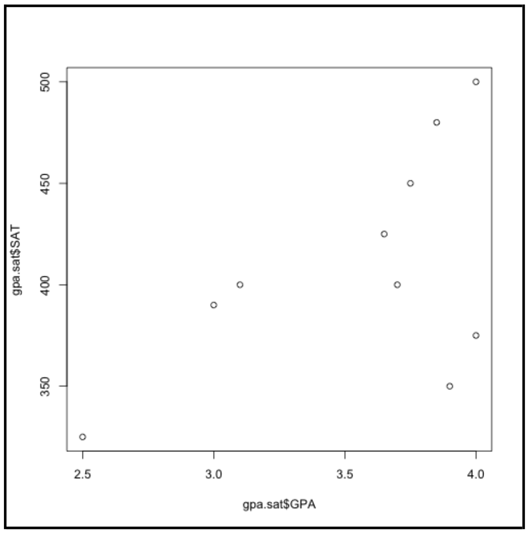
To determine if two quantitative variables are related to one another, we can calculate the correlation between the two variables. We can also create an equation that can be used to predict one of these quantitative variables from another. Here, we will focus on the variables GPA and SAT score.


```
> gpa.sat<-read.csv(file.choose(),header=T)
> gpa.sat
    GPA SAT
1  3.85 480
2  3.65 425
3  3.70 400
4  4.00 500
5  3.00 390
6  3.90 350
7  4.00 375
8  2.50 325
9  3.10 400
10 3.75 450
```

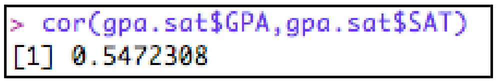
It is important to get into the habit of creating a scatterplot before computing the correlation coefficient and obtaining the regression equation to make sure the relationship between the variables in question is linear. The line of code below is the minimum amount of syntax needed for creating a scatterplot (e.g., plot(x,y)).

```
plot(x=gpa.sat$GPA,y=gpa.sat$SAT)
```

If you type in this code, you will get the following scatterplot.



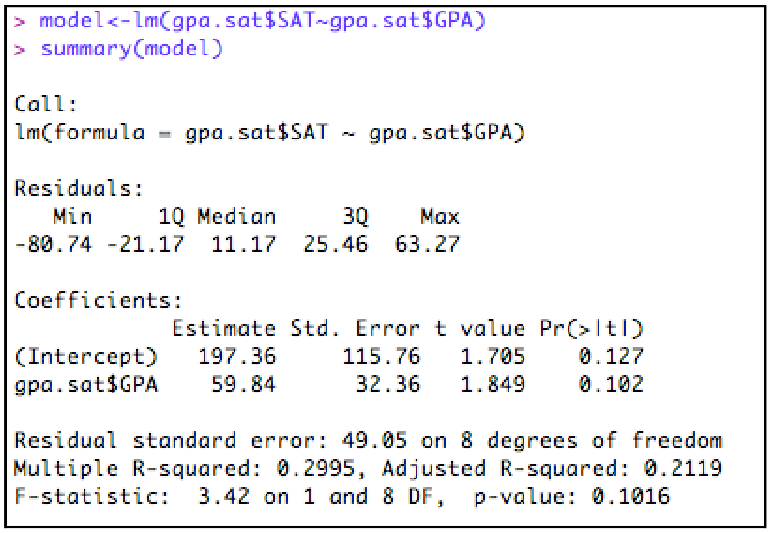
To obtain the correlation between GPA and SAT score, use the `cor()` function as shown below.



```
> cor(gpa.sat$GPA,gpa.sat$SAT)
[1] 0.5472308
```

The arguments for the `cor()` function are the two variables you are interested in correlating. The number above, 0.5472308, is the value of r, the Pearson Product- Moment Correlation Coefficient.

The `lm()` function will be used to get the regression equation that can be used to predict SAT score from GPA. Format: `lm(y~x)`.



```
> model<-lm(gpa.sat$SAT~gpa.sat$GPA)
> summary(model)

Call:
lm(formula = gpa.sat$SAT ~ gpa.sat$GPA)

Residuals:
   Min     1Q Median     3Q    Max
-80.74 -21.17  11.17  25.46  63.27

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   197.36     115.76   1.705    0.127
gpa.sat$GPA    59.84      32.36   1.849    0.102

Residual standard error: 49.05 on 8 degrees of freedom
Multiple R-squared: 0.2995, Adjusted R-squared: 0.2119
F-statistic:  3.42 on 1 and 8 DF,  p-value: 0.1016
```

You should to save your `lm()` function under an object name as you will be needing it to extract additional output using the `summary()` function. As a guide for writing the function, you should type lm(y-variable ~ x-variable). So in the example above, our y-variable is `SAT` from the `gpa.sat` data set and our x-variable is `GPA` from the `gpa.sat` data set. Here, we labeled the `lm()` object `model`.

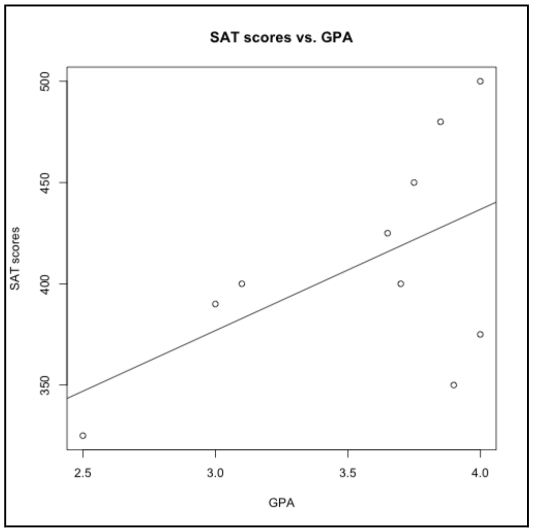
To obtain the regression equation information, you need to use the `summary()` function on the `lm()` object name, so for the example above, the syntax is `summary(model)`. The predicted equation from the output above is:

Predicted SAT = 197.36 + 59.84*GPA

You can find the regression coefficients under the **Coefficients** section of the output and then under the column labeled **Estimate**. The estimated intercept value is the first row and the estimated slope value is in the second row.

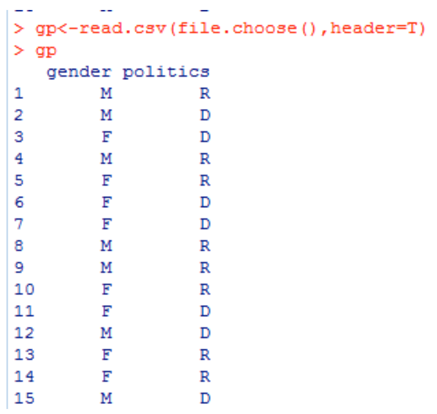
Finally, you can use the model object to plot the predicted regression line in the plot window (when the plot window contains a scatterplot of the two variables). You do this by typing the following command at the command prompt:

```
abline(model)
```



Chi-square Test of Independence

To determine how two categorical variables relate (like gender and political party affiliation), we can use the chi-square test of independence. Here, we have a sample of males and females, and each male (M) and female (F) is categorized as being either Republican (R) or Democrat (D). Is there a significant relationship between gender and politics? To begin, we will first read in a .csv file and we'll name our object **gp** (for Gender Politics). As you see below, once we read in the file and then type our object name, we can see the data. We have two columns in our data set corresponding to **gender** and **politics**.



```
> gp<-read.csv(file.choose(),header=T)
> gp
   gender politics
1       M        R
2       M        D
3       F        D
4       M        R
5       F        R
6       F        D
7       F        D
8       M        R
9       M        R
10      F        R
11      F        D
12      M        D
13      F        R
14      F        R
15      M        D
```

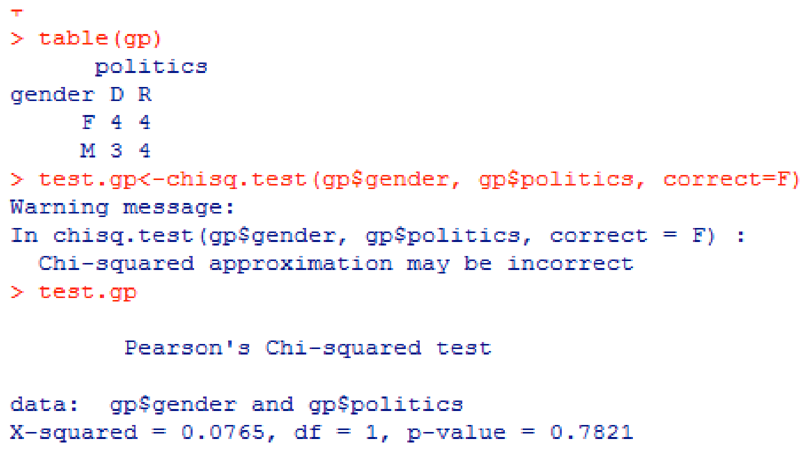
We can next create a contingency table by typing in table (gp).

To conduct a chi-square test, we now have to name a new object, and we'll call this test.gp. As you see below, you'll need to type in the following command:

```
test.gp<-chisq.test(gp$gender, gp$politics, correct=F)
```

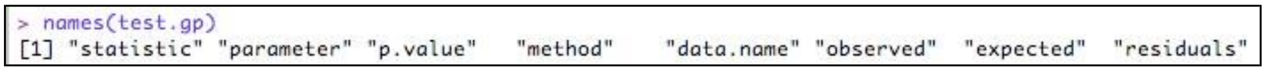
We type in `correct = F` because without this, R automatically conducts a more conservative chi-square analysis by applying something called the Yates' Continuity Correction. We don't want that here.

Note when you hit enter after typing the above command, you will get a warning message. Please proceed despite this message by typing the new object name (test.gp) to get the chi-square results.



```
> table(gp)
      politics
gender D R
     F 4 4
     M 3 4
> test.gp<-chisq.test(gp$gender, gp$politics, correct=F)
Warning message:
In chisq.test(gp$gender, gp$politics, correct = F) :
  Chi-squared approximation may be incorrect
> test.gp

        Pearson's Chi-squared test

data:  gp$gender and gp$politics
X-squared = 0.0765, df = 1, p-value = 0.7821
```

There is additional information that `chisq.test()` does not output automatically. To see what else is stored in `test.gp`, type `names(test.gp)`.



```
> names(test.gp)
[1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"  "expected"  "residuals"
```

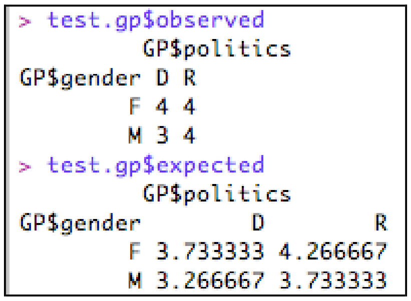
The test automatically outputs the following when you type in the `chisq.test()` object name, `test.gp`:

- `statistic`: the value the chi-squared test statistic.
- `parameter`: the degrees of freedom of the approximate chi-squared distribution of the test statistic
- `p.value`: the *p*-value for the test.
- `method`: a character string indicating the type of test performed, and whether Monte Carlo simulation or continuity correction was used.
- `data.name`: a character string giving the name(s) of the data.

The three options that are embedded within the function are

- `observed`: the observed counts.
- `expected`: the expected counts under the null hypothesis.
- `residuals`: the Pearson residuals, (observed - expected) / sqrt(expected).

We can access those options by typing in the (chisq.test object name)$(option name). So, if you want, for example, to see the observed counts and the expected counts (to compare them), you could do the following:



```
> test.gp$observed
         GP$politics
GP$gender D R
        F 4 4
        M 3 4
> test.gp$expected
         GP$politics
GP$gender        D        R
        F 3.733333 4.266667
        M 3.266667 3.733333
```

Now, with the above example, you began with a data file that included columns for gender and politics. What if, instead of having that data file, you were given a contingency table like the one below? Here, again, we have the same sample of males and females, and each male and female is categorized as being either Republican (r) or Democrat (d). Is there a significant relationship between gender and political party affiliation?

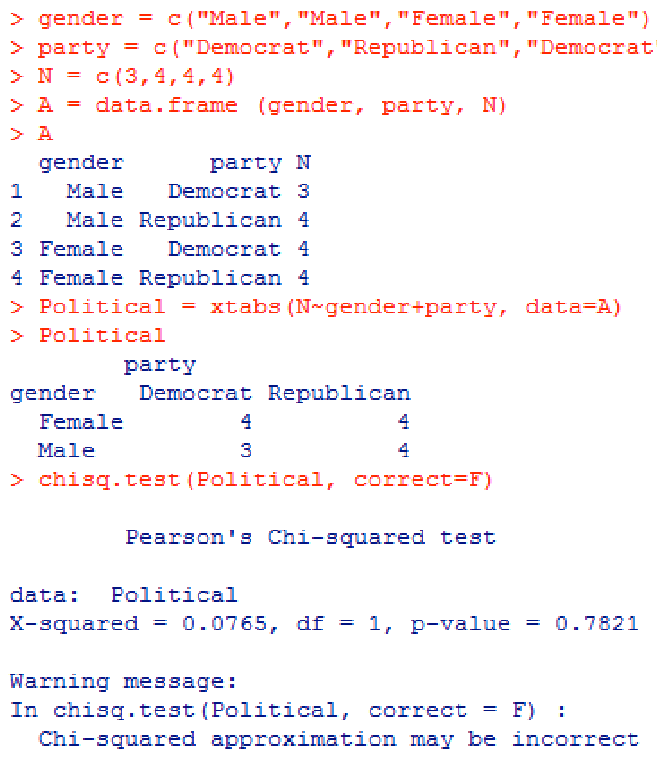
	Males	Females	Totals
Democrat	3	4	7
Republican	4	4	8
Totals	7	8	15

We can enter the above information rather easily right into R in order to perform a chi-square analysis.

We begin by defining our variables (gender and party). Note we want to enter the information in a particular way here (we'll first enter the two values for males (and we'll begin with the Democrat value) and then we'll enter the two values for females).

We then define a data frame (which you could call anything; I called it "A" in the example below).

Next, we name our contingency table. Here, I called that Political. I then asked to perform a chi-square test of Political. Note the results are just the same as they were above when we began with an original data file (rather than a contingency table). We just did things in a slightly different way.



```
> gender = c("Male","Male","Female","Female")
> party = c("Democrat","Republican","Democrat","Republican")
> N = c(3,4,4,4)
> A = data.frame (gender, party, N)
> A
  gender      party N
1   Male   Democrat 3
2   Male Republican 4
3 Female   Democrat 4
4 Female Republican 4
> Political = xtabs(N~gender+party, data=A)
> Political
        party
gender   Democrat Republican
  Female        4          4
  Male          3          4
> chisq.test(Political, correct=F)

        Pearson's Chi-squared test

data:  Political
X-squared = 0.0765, df = 1, p-value = 0.7821

Warning message:
In chisq.test(Political, correct = F) :
  Chi-squared approximation may be incorrect
```

In terms of the warning message, you can see that we get a message that the chi-square approximation may be incorrect. This is very likely because of the small expected values. If you want to see the expected values, remember that you can do the following. This involves first creating an object that holds the chi-square results.

```
> test.political<-chisq.test(Political, correct=F)
Warning message:
In chisq.test(Political, correct = F) :
  Chi-squared approximation may be incorrect
> test.political

        Pearson's Chi-squared test

data:  Political
X-squared = 0.0765, df = 1, p-value = 0.7821

> test.political$expected
        party
gender   Democrat Republican
  Female 3.733333   4.266667
  Male   3.266667   3.733333
```