# Tutorial for the R Statistical Package

University of Colorado Denver

Stephanie Santorico
Mark Shin

## Contents

**1 Basics** **2**

**2 Importing Data** **10**

**3 Basic Analysis** **14**

**4 Plotting** **22**

**5 Installing Packages** **29**

This document is associated with datasets and scripts as well as a tutorial video available at http://math.ucdenver.edu/RTutorial

Last Updated January 2010

# 1 Basics

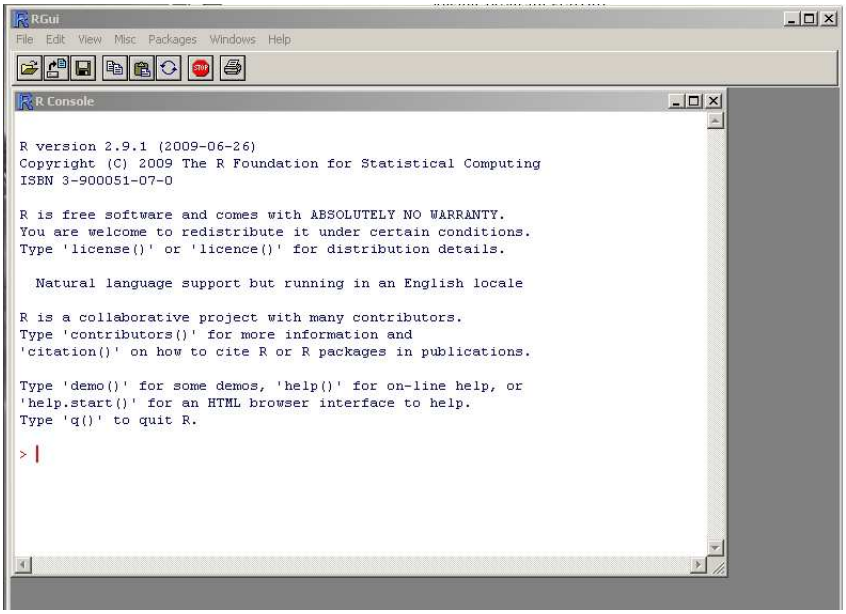
This tutorial will look at the open source statistical software package R. Students that are not familiar with command line operations may feel intimidated by the way a user interacts with R, but this tutorial series should alleviate these feelings and help lessen the learning curve of this software.

Why should I use R for my work? R has many benefits over other statistical software packages. Its main benefit is that it is open source software. This means that anyone can help develop new packages and features. This allows cutting edge methods to come to R much faster than other software packages. This software can also be downloaded for free by anyone from various sites around the world.

Let's first start by downloading and installing R on your machine. After connecting to the internet go to: www.r-project.org. From here select CRAN from the menu on the left side of the page. On the next page select a mirror from which to download the package from. The mirrors in the United States are near the bottom of the page. From this point you need to select the operating system type you are using (e.g. Mac or Windows). From here either select the "base" option for Windows machines or the current dmg file for Mac computers (currently R-2.10.1.dgm). These selections will download the proper software for your computer.

Now double click on the file you downloaded and follow the instructions to complete your installation.

Now let's start using R. Every time you start up R you should see the following program startup:



The last line of the output includes the > prompt. This is where you enter in commands that tell R what you want it to do. From here you use commands to do everything from reading in your data to creating graphs to carrying out statistical tests.

Let's start by exploring how R is designed. Everything in R from your dataset to summaries of statistical tests are classified as objects. We'll look first at an example to show you what we mean by this.

We can create a vector in R consisting of the numbers from one to ten and name it v by entering the following command after the > prompt:

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

The assignment operator in R is <-. The c used in the command stands for concatenate. Here we are concatenating, or binding together the numbers from one to ten and storing them as v. Now let's call our vector back and see what is in it. By typing v back into the command line we can recall v.

```
> v
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

Now let's say we only want to call the 7th object in our vector. For any one dimensional object in R, we can access a specific entry by using hard brackets after the object. In our current situation if we want to select the 7th object we simply type:

```
> v[7]
[1] 7
>
```

When we load our data sets into R, they are classified as a different type of object called a data frame. Data frames can have names associated with the variables but don't have to. Let's look at a preloaded dataset in R to see how it looks. The dataset we will look at here is called iris. Type iris on the command line and hit enter.

```
 > iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1           5.1         3.5          1.4         0.2     setosa
2           4.9         3.0          1.4         0.2     setosa
3           4.7         3.2          1.3         0.2     setosa
4           4.6         3.1          1.5         0.2     setosa
5           5.0         3.6          1.4         0.2     setosa
6           5.4         3.9          1.7         0.4     setosa
7           4.6         3.4          1.4         0.3     setosa
8           5.0         3.4          1.5         0.2     setosa
9           4.4         2.9          1.4         0.2     setosa
```

| | | | | | |
|---|---|---|---|---|---|
| 10 | 4.9 | 3.1 | 1.5 | 0.1 | setosa |
| 11 | 5.4 | 3.7 | 1.5 | 0.2 | setosa |
| 12 | 4.8 | 3.4 | 1.6 | 0.2 | setosa |
| 13 | 4.8 | 3.0 | 1.4 | 0.1 | setosa |
| 14 | 4.3 | 3.0 | 1.1 | 0.1 | setosa |
| 15 | 5.8 | 4.0 | 1.2 | 0.2 | setosa |
| 16 | 5.7 | 4.4 | 1.5 | 0.4 | setosa |
| 17 | 5.4 | 3.9 | 1.3 | 0.4 | setosa |
| 18 | 5.1 | 3.5 | 1.4 | 0.3 | setosa |
| 19 | 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 20 | 5.1 | 3.8 | 1.5 | 0.3 | setosa |
| 21 | 5.4 | 3.4 | 1.7 | 0.2 | setosa |
| 22 | 5.1 | 3.7 | 1.5 | 0.4 | setosa |
| 23 | 4.6 | 3.6 | 1.0 | 0.2 | setosa |
| 24 | 5.1 | 3.3 | 1.7 | 0.5 | setosa |
| 25 | 4.8 | 3.4 | 1.9 | 0.2 | setosa |
| 26 | 5.0 | 3.0 | 1.6 | 0.2 | setosa |
| 27 | 5.0 | 3.4 | 1.6 | 0.4 | setosa |
| 28 | 5.2 | 3.5 | 1.5 | 0.2 | setosa |
| 29 | 5.2 | 3.4 | 1.4 | 0.2 | setosa |
| 30 | 4.7 | 3.2 | 1.6 | 0.2 | setosa |
| 31 | 4.8 | 3.1 | 1.6 | 0.2 | setosa |
| 32 | 5.4 | 3.4 | 1.5 | 0.4 | setosa |
| 33 | 5.2 | 4.1 | 1.5 | 0.1 | setosa |
| 34 | 5.5 | 4.2 | 1.4 | 0.2 | setosa |
| 35 | 4.9 | 3.1 | 1.5 | 0.2 | setosa |
| 36 | 5.0 | 3.2 | 1.2 | 0.2 | setosa |
| 37 | 5.5 | 3.5 | 1.3 | 0.2 | setosa |
| 38 | 4.9 | 3.6 | 1.4 | 0.1 | setosa |
| 39 | 4.4 | 3.0 | 1.3 | 0.2 | setosa |
| 40 | 5.1 | 3.4 | 1.5 | 0.2 | setosa |
| 41 | 5.0 | 3.5 | 1.3 | 0.3 | setosa |
| 42 | 4.5 | 2.3 | 1.3 | 0.3 | setosa |
| 43 | 4.4 | 3.2 | 1.3 | 0.2 | setosa |
| 44 | 5.0 | 3.5 | 1.6 | 0.6 | setosa |
| 45 | 5.1 | 3.8 | 1.9 | 0.4 | setosa |
| 46 | 4.8 | 3.0 | 1.4 | 0.3 | setosa |
| 47 | 5.1 | 3.8 | 1.6 | 0.2 | setosa |
| 48 | 4.6 | 3.2 | 1.4 | 0.2 | setosa |
| 49 | 5.3 | 3.7 | 1.5 | 0.2 | setosa |
| 50 | 5.0 | 3.3 | 1.4 | 0.2 | setosa |
| 51 | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| 52 | 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| 53 | 6.9 | 3.1 | 4.9 | 1.5 | versicolor |
| 54 | 5.5 | 2.3 | 4.0 | 1.3 | versicolor |
| 55 | 6.5 | 2.8 | 4.6 | 1.5 | versicolor |
| 56 | 5.7 | 2.8 | 4.5 | 1.3 | versicolor |
| 57 | 6.3 | 3.3 | 4.7 | 1.6 | versicolor |
| 58 | 4.9 | 2.4 | 3.3 | 1.0 | versicolor |
| 59 | 6.6 | 2.9 | 4.6 | 1.3 | versicolor |
| 60 | 5.2 | 2.7 | 3.9 | 1.4 | versicolor |
| 61 | 5.0 | 2.0 | 3.5 | 1.0 | versicolor |
| 62 | 5.9 | 3.0 | 4.2 | 1.5 | versicolor |
| 63 | 6.0 | 2.2 | 4.0 | 1.0 | versicolor |

| | | | | | |
|---|---|---|---|---|---|
| 64 | 6.1 | 2.9 | 4.7 | 1.4 | versicolor |
| 65 | 5.6 | 2.9 | 3.6 | 1.3 | versicolor |
| 66 | 6.7 | 3.1 | 4.4 | 1.4 | versicolor |
| 67 | 5.6 | 3.0 | 4.5 | 1.5 | versicolor |
| 68 | 5.8 | 2.7 | 4.1 | 1.0 | versicolor |
| 69 | 6.2 | 2.2 | 4.5 | 1.5 | versicolor |
| 70 | 5.6 | 2.5 | 3.9 | 1.1 | versicolor |
| 71 | 5.9 | 3.2 | 4.8 | 1.8 | versicolor |
| 72 | 6.1 | 2.8 | 4.0 | 1.3 | versicolor |
| 73 | 6.3 | 2.5 | 4.9 | 1.5 | versicolor |
| 74 | 6.1 | 2.8 | 4.7 | 1.2 | versicolor |
| 75 | 6.4 | 2.9 | 4.3 | 1.3 | versicolor |
| 76 | 6.6 | 3.0 | 4.4 | 1.4 | versicolor |
| 77 | 6.8 | 2.8 | 4.8 | 1.4 | versicolor |
| 78 | 6.7 | 3.0 | 5.0 | 1.7 | versicolor |
| 79 | 6.0 | 2.9 | 4.5 | 1.5 | versicolor |
| 80 | 5.7 | 2.6 | 3.5 | 1.0 | versicolor |
| 81 | 5.5 | 2.4 | 3.8 | 1.1 | versicolor |
| 82 | 5.5 | 2.4 | 3.7 | 1.0 | versicolor |
| 83 | 5.8 | 2.7 | 3.9 | 1.2 | versicolor |
| 84 | 6.0 | 2.7 | 5.1 | 1.6 | versicolor |
| 85 | 5.4 | 3.0 | 4.5 | 1.5 | versicolor |
| 86 | 6.0 | 3.4 | 4.5 | 1.6 | versicolor |
| 87 | 6.7 | 3.1 | 4.7 | 1.5 | versicolor |
| 88 | 6.3 | 2.3 | 4.4 | 1.3 | versicolor |
| 89 | 5.6 | 3.0 | 4.1 | 1.3 | versicolor |
| 90 | 5.5 | 2.5 | 4.0 | 1.3 | versicolor |
| 91 | 5.5 | 2.6 | 4.4 | 1.2 | versicolor |
| 92 | 6.1 | 3.0 | 4.6 | 1.4 | versicolor |
| 93 | 5.8 | 2.6 | 4.0 | 1.2 | versicolor |
| 94 | 5.0 | 2.3 | 3.3 | 1.0 | versicolor |
| 95 | 5.6 | 2.7 | 4.2 | 1.3 | versicolor |
| 96 | 5.7 | 3.0 | 4.2 | 1.2 | versicolor |
| 97 | 5.7 | 2.9 | 4.2 | 1.3 | versicolor |
| 98 | 6.2 | 2.9 | 4.3 | 1.3 | versicolor |
| 99 | 5.1 | 2.5 | 3.0 | 1.1 | versicolor |
| 100 | 5.7 | 2.8 | 4.1 | 1.3 | versicolor |
| 101 | 6.3 | 3.3 | 6.0 | 2.5 | virginica |
| 102 | 5.8 | 2.7 | 5.1 | 1.9 | virginica |
| 103 | 7.1 | 3.0 | 5.9 | 2.1 | virginica |
| 104 | 6.3 | 2.9 | 5.6 | 1.8 | virginica |
| 105 | 6.5 | 3.0 | 5.8 | 2.2 | virginica |
| 106 | 7.6 | 3.0 | 6.6 | 2.1 | virginica |
| 107 | 4.9 | 2.5 | 4.5 | 1.7 | virginica |
| 108 | 7.3 | 2.9 | 6.3 | 1.8 | virginica |
| 109 | 6.7 | 2.5 | 5.8 | 1.8 | virginica |
| 110 | 7.2 | 3.6 | 6.1 | 2.5 | virginica |
| 111 | 6.5 | 3.2 | 5.1 | 2.0 | virginica |
| 112 | 6.4 | 2.7 | 5.3 | 1.9 | virginica |
| 113 | 6.8 | 3.0 | 5.5 | 2.1 | virginica |
| 114 | 5.7 | 2.5 | 5.0 | 2.0 | virginica |
| 115 | 5.8 | 2.8 | 5.1 | 2.4 | virginica |
| 116 | 6.4 | 3.2 | 5.3 | 2.3 | virginica |
| 117 | 6.5 | 3.0 | 5.5 | 1.8 | virginica |

```
118          7.7         3.8          6.7         2.2  virginica
119          7.7         2.6          6.9         2.3  virginica
120          6.0         2.2          5.0         1.5  virginica
121          6.9         3.2          5.7         2.3  virginica
122          5.6         2.8          4.9         2.0  virginica
123          7.7         2.8          6.7         2.0  virginica
124          6.3         2.7          4.9         1.8  virginica
125          6.7         3.3          5.7         2.1  virginica
126          7.2         3.2          6.0         1.8  virginica
127          6.2         2.8          4.8         1.8  virginica
128          6.1         3.0          4.9         1.8  virginica
129          6.4         2.8          5.6         2.1  virginica
130          7.2         3.0          5.8         1.6  virginica
131          7.4         2.8          6.1         1.9  virginica
132          7.9         3.8          6.4         2.0  virginica
133          6.4         2.8          5.6         2.2  virginica
134          6.3         2.8          5.1         1.5  virginica
135          6.1         2.6          5.6         1.4  virginica
136          7.7         3.0          6.1         2.3  virginica
137          6.3         3.4          5.6         2.4  virginica
138          6.4         3.1          5.5         1.8  virginica
139          6.0         3.0          4.8         1.8  virginica
140          6.9         3.1          5.4         2.1  virginica
141          6.7         3.1          5.6         2.4  virginica
142          6.9         3.1          5.1         2.3  virginica
143          5.8         2.7          5.1         1.9  virginica
144          6.8         3.2          5.9         2.3  virginica
145          6.7         3.3          5.7         2.5  virginica
146          6.7         3.0          5.2         2.3  virginica
147          6.3         2.5          5.0         1.9  virginica
148          6.5         3.0          5.2         2.0  virginica
149          6.2         3.4          5.4         2.3  virginica
150          5.9         3.0          5.1         1.8  virginica
>
```

The left column displays the observation number in the dataset. The remaining data in the five columns on the right are the actual data in the dataset. You can also see from this example that you don't have to just store numbers in your datasets. The 5th column in the dataset uses text to differentiate the type of species of iris being observed in this study.

Now with two dimensional objects, like this dataset, if you want to call a specific entry you must specify the row and column in the object [r,c] with r being the row and c the column in the object you wish to select. Let's use the technique with this data frame. Suppose we want to see what value is in the 3rd column in the first row.

```
> iris[1,3]
[1] 1.4
>
```

How about the 5th column in the 100th row?

```
> iris[100,5]
[1] versicolor
Levels: setosa versicolor virginica
>
```

This output is letting us know that the 100th row and 5th column is versicolor. Since it is not a numerical value, R also displays the other two values for this variable, setosa and virginica.

In R it is very important to make sure you spell the names of your datasets and commands correctly and with the same capitalization as you originally set them up. Try typing Iris into your command line.

```
> Iris
Error: object "Iris" not found
>
```

As you can see R doesn't know what we are trying to call when we type in the capitalized version.

Now let's go back to our vector example. We easily created a string of ten one dimensional vectors by separating them by commas. But what if we wanted to create a vector of one hundred numbers, or one thousand? Typing each one out would be very time consuming and very inefficient. Luckily R has easier ways to accomplish this.

```
> v2 <- c(1:10)
> v2
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

The : operator tells R that I want a string of numbers from one to 10 counting by ones. Another way to do this is the seq() command.

```
> v3 <- seq(1,10)
> v3
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

These commands are very useful when using R. For instance suppose we want to only look at the first 10 entries in our iris data set. Remembering that R asks for rows first and then columns:

```
> iris[1:10,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
```

```
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
>
```

By leaving the columns position blank, I am telling R that I want all columns associated with iris but only the first 10 rows. We can even make these first 10 rows of the iris dataset into its own object, let's call it `iris10`.

```
> iris10 <- iris[1:10,]
>
```

It's very common to create many of these subsets and strings when you are analyzing your data. R will not delete any of them as you complete your analysis, but if you name multiple objects the same name, it will overwrite them. For instance, call your `iris10` object again.

```
> iris10
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
>
```

Now let's name a string of vectors `iris10`:

```
> iris10 <- c(1:10)
> iris10
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

You can see that the object named `iris10` is now associated with the string of numbers and not the first 10 rows of the iris dataset. Also note that R did not warn you that you were overwriting the original `iris10` variable, so you must be careful when naming your objects.

What if we want to see all the objects that are available to us? The command ls() will list all the objects that you have created or read into R.

```
> ls()
[1] "iris10" "v"       "v2"      "v3"
>
```

Note that because iris is a special pre-loaded dataset it was not in the list. Let's suppose that we want to remove the `iris10` vector from the objects we have to work with. The command to delete objects is rm().

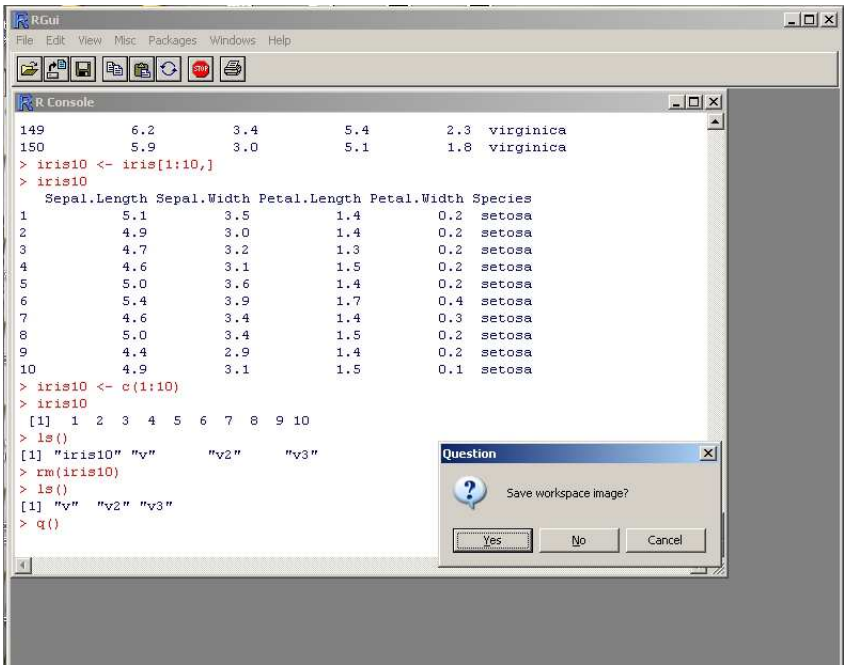
```
> rm(iris10)
>
```

Now let's list the objects in our workspace again.

```
> ls()
[1] "v"   "v2" "v3"
>
```

You can see that `iris10` is no longer available on our workspace.

I think this is a good stopping point for now. To quit R, simply type q() into the command prompt.



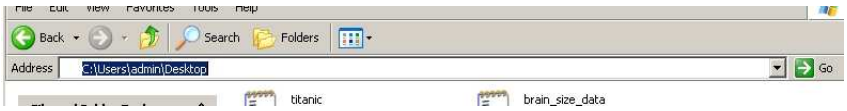

Now upon exiting R it will ask you if you would like to save your workspace. If you select Yes it will keep all objects that were listed when you executed the ls() command. If you select No, the next time you run R you will be starting over with all your objects, datasets and variables. There may be times when either choice is preferable and you can always check to see what objects are on your workspace with the ls() command when you start R up again.

This first lesson demonstrates some basic operations in R. We'll next look at how to load external datasets and how to do some basic analyses.

# 2 Importing Data

Now that we've covered some basics of how R works and are more comfortable with the command line, let's move on to looking at how to load data into R.

Again you will be using the command line prompt in R to tell the software what you want it to do, this includes reading in your datasets. Let's start by downloading our dataset titled `cancer_and_smoking.txt` from our tutorial website. I'm going to save my dataset on the "desktop" on my computer. Now, when you open a Microsoft Word document on your computer you have to show Microsoft Office where your Word file is located on your computer by selecting the correct folder and location. The same is true in R. Instead of clicking on the folder icons like in MS Office, R asks you to tell it through the command line. For my computer configuration, the location of my dataset is `C:\Documents_and_Settings\Admin\Desktop\cancer_and_smoking.csv`. Depending on your computer this location may vary. The easiest way to determine your dataset's location is to look at the navigation bar at the top of the window of the folder that contains your dataset. This tool bar should look something like this:




The type of data file we will start with is a csv file. These files can be read into Microsoft Excel easily and are very commonly used. The command in R to read these files is read.csv(). Inside the parentheses we need to tell R a few things about our data, most importantly where it is located. R requires you to have two sets of back slashes between folders in the command when calling the read.csv command. In this situation the command to read in my data will look like this:

```
>smoking <- read.csv("C:\\Documents and Settings\\Admin\\Desktop
\\cancer_and_smoking.csv", header = F)
>
```

Now let's look at what we just read in. Remember we can print out our smoking dataset by typing smoking into the command prompt, let's look at the first 10 entries of this dataset.

```
> smoking[1:10,]
1     AL 18.20 2.90 17.05 1.59 6.15
2     AZ 25.82 3.52 19.80 2.75 6.61
3     AR 18.24 2.99 15.98 2.02 6.94
4     CA 28.60 4.46 22.07 2.66 7.06
5     CT 31.10 5.11 22.83 3.35 7.20
6     DE 33.60 4.78 24.55 3.36 6.45
7     DC 40.46 5.60 27.27 3.13 7.08
```

```
8     FL 28.27 4.46 23.57 2.41 6.07
9     ID 20.10 3.08 13.58 2.46 6.62
10    IL 27.91 4.75 22.80 2.95 7.27
>
```

As you can see, this dataset shows some data by state. The headers for each data column should be State, Cigarette, Bladder, Lung, Kidney, and Leukemia. Unfortunately this dataset did not have the data headers built in but we can always add them. Recall our concatenate function, c(), that we learned in the first lesson. This function works with things other than numbers. To add headers to our data, we call the following command:

```
> names(smoking) <- c("State", "Cigarette", "Bladder", "Lung",
  "Kidney", "Leukemia")
>
```

Now let's call our dataset again,

```
> smoking[1:10,]
   State   Cigarette Bladder  Lung  Kidney Leukemia
1     AL 18.20 2.90 17.05 1.59 6.15
2     AZ 25.82 3.52 19.80 2.75 6.61
3     AR 18.24 2.99 15.98 2.02 6.94
4     CA 28.60 4.46 22.07 2.66 7.06
5     CT 31.10 5.11 22.83 3.35 7.20
6     DE 33.60 4.78 24.55 3.36 6.45
7     DC 40.46 5.60 27.27 3.13 7.08
8     FL 28.27 4.46 23.57 2.41 6.07
9     ID 20.10 3.08 13.58 2.46 6.62
10    IL 27.91 4.75 22.80 2.95 7.27
>
```

We now have names associated with each column. This makes it much easier to interoperate and call other functions in R to analyze the dataset.

R can read many different types of dataset formats. For all dataset formats, including .txt files, you can call the read.table() command, you just have to tell R a few more details about the dataset format.

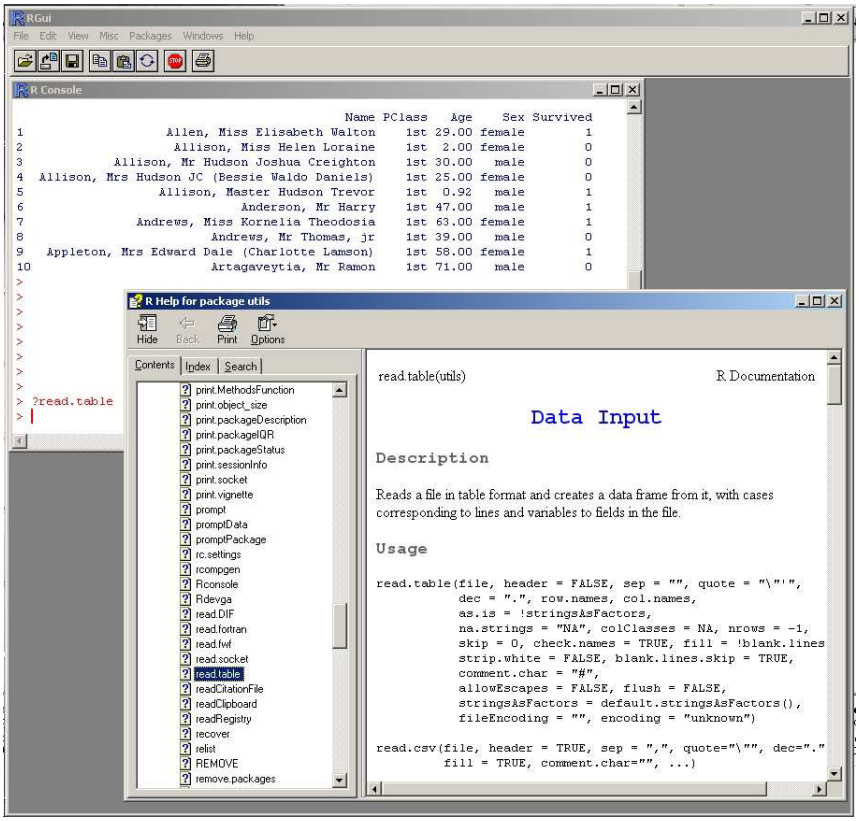
Let's try using some of these other functions on a new dataset. We'll next look at the dataset titanic.txt. This dataset does include the data headers in the file. We need to tell R that the first row does contain the headers and not data. To do this we will add an additional option to the read.table command. Additionally, we must tell R how the data is formatted in the text file. Are the data columns separated by a comma, a space or tab? In this case it is separated by a tab so we must include that in the read.table() command as well:

```
> titanic <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
titanic.txt", header=T, sep="\t")
>
```

Now let's check what we added into the command. The first command in the read.table() function is the location of the dataset. The second statement tells R that there is a header in the file for the variables (header = F if there is no header). The next statement tells R that the dataset file has columns separated by a tab. Now let's look at the first few lines of this dataset:

```
> titanic[1:10,]
                                                 Name PClass    Age     Sex Survived
1                       Allen, Miss Elisabeth Walton    1st 29.00 female        1
2                        Allison, Miss Helen Loraine    1st  2.00 female        0
3                Allison, Mr Hudson Joshua Creighton    1st 30.00   male        0
4  Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female        0
5                      Allison, Master Hudson Trevor    1st  0.92   male        1
6                                 Anderson, Mr Harry    1st 47.00   male        1
7                   Andrews, Miss Kornelia Theodosia    1st 63.00 female        1
8                             Andrews, Mr Thomas, jr    1st 39.00   male        0
9    Appleton, Mrs Edward Dale (Charlotte Lamson)    1st 58.00 female        1
10                            Artagaveytia, Mr Ramon    1st 71.00   male        0
>
```

This may sound pretty complex. With all these different data types and different scenarios, what do I need to include? All R commands will have corresponding help files. The easiest way to ask R for help is to type the ? followed by the command name. The other way to call help in R is the help() command. Let's try this for our read.table() command.



As you can see, a new window will pop up that shows help for the command. It will break down all the options for the command, their default values and descriptions of the options. This is always a great first step if you run into problems with any R function.

In this lesson we have demonstrated how to import two very common formats of data into R, how to associate the variable names and to find additional help for functions in R. In the next lesson we will begin some basic analysis on our datasets and learn how to manipulate data in R.

# 3 Basic Analysis

Now that we have read in our dataset, let's look at some of the characteristics of it. It's always a good idea to take a look at what you have just read into R to make sure it was done correctly. I always like to look at the first few observations (like we did with the titanic data above). Also, with a few simple commands, R will provide a few very helpful characteristics like the header names, the size of your dataset and a table that can describe your dataset.

Let's take a look at the titanic dataset.

```
> titanic <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
titanic.txt", header=T, sep="\t")
> names(titanic)
[1] "Name"     "PClass"   "Age"      "Sex"      "Survived"
> dim(titanic)
[1] 1313    5
> table(titanic$Survived)

  0   1
863 450
>
```

As you can see, these commands create an initial overview of your dataset and make it easy to see if you read it into R correctly. The table() command tabulates the data for one of your discrete variables. In this case I asked it to tell me the number of people in the dataset that survived and did not survive. Notice that the dollar symbol, $, tells R that I am looking for a table of the titanic object with regard to the variable Survived.

Now let's suppose that I want to create a new data frame that contains all the people in the titanic dataset that survived. Since the dataset has a variable coded as 1 for Survived, it is easy to subset our original dataset with the following command:

```
titanic.survived <- titanic[titanic$Survived == 1, ]
> titanic.survived[1:10,]
                                                 Name PClass   Age    Sex
1                       Allen, Miss Elisabeth Walton    1st 29.00 female
5                      Allison, Master Hudson Trevor    1st  0.92   male
6                                  Anderson, Mr Harry    1st 47.00   male
7                  Andrews, Miss Kornelia Theodosia    1st 63.00 female
9      Appleton, Mrs Edward Dale (Charlotte Lamson)    1st 58.00 female
12 Astor, Mrs John Jacob (Madeleine Talmadge Force)    1st 19.00 female
13                      Aubert, Mrs Leontine Pauline    1st    NA female
14                          Barkworth, Mr Algernon H    1st    NA   male
16   Baxter, Mrs James (Helene DeLaudeniere Chaput)    1st 50.00 female
19                      Beckwith, Mr Richard Leonard    1st 37.00   male
   Survived
1         1
5         1
6         1
```

```
7         1
9         1
12        1
13        1
14        1
16        1
19        1
>
```

As we can see from the output, only people that survived are included in the titanic.survived data frame. Note that this command does not move these entries in our original dataset to this new data frame, but it does copy them.

Let's now look at another very useful feature of R. Looking back at the smoking and cancer dataset, suppose that we have come up with a new metric to measure cancer by adding up bladder cancer, lung cancer, kidney cancer and leukemia rates. This is very simple to do by hand or even in an Excel spreadsheet. But we can do this on the fly and create a new variable very quickly in R and even attach it to our dataset.

```
> smoking <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
cancer_and_smoking.txt", header=T, sep="\t")
> names(smoking)
[1] "STATE"     "CIG"      "BLAD"     "LUNG"     "KID"      "LEUK"
> RATE <- smoking$BLAD + smoking$LUNG + smoking$KID + smoking$LEUK
> RATE
 [1] 27.69 32.68 27.93 36.25 38.49 39.14 43.08 36.51 25.74 37.77 34.20 31.41
[13] 30.05 29.11 39.11 35.19 41.35 36.65 37.87 29.74 26.51 34.39 33.78 31.14
[25] 39.09 42.17 25.96 40.65 25.62 36.69 32.29 26.58 37.86 28.57 29.01 29.31
[37] 33.66 24.23 35.58 34.64 34.76 30.97 27.56 38.56
>
```

This new variable is called RATE, and our command creates a string that sums up the bladder cancer, lung cancer, kidney cancer and leukemia rates. Now if we wish to attach it to our dataset we simply have to add this column to the data frame. The command to do this is called cbind().

```
> smoking <- cbind(smoking, RATE)
> smoking
   STATE   CIG BLAD  LUNG  KID LEUK  RATE
1     AL 18.20 2.90 17.05 1.59 6.15 27.69
2     AZ 25.82 3.52 19.80 2.75 6.61 32.68
3     AR 18.24 2.99 15.98 2.02 6.94 27.93
4     CA 28.60 4.46 22.07 2.66 7.06 36.25
5     CT 31.10 5.11 22.83 3.35 7.20 38.49
6     DE 33.60 4.78 24.55 3.36 6.45 39.14
7     DC 40.46 5.60 27.27 3.13 7.08 43.08
8     FL 28.27 4.46 23.57 2.41 6.07 36.51
9     ID 20.10 3.08 13.58 2.46 6.62 25.74
10    IL 27.91 4.75 22.80 2.95 7.27 37.77
11    IN 26.18 4.09 20.30 2.81 7.00 34.20
12    IO 22.12 4.23 16.59 2.90 7.69 31.41
13    KS 21.84 2.91 16.84 2.88 7.42 30.05
```

```
14    KY 23.44 2.86 17.71 2.13 6.41 29.11
15    LA 21.58 4.65 25.45 2.30 6.71 39.11
16    ME 28.92 4.79 20.94 3.22 6.24 35.19
17    MD 25.91 5.21 26.48 2.85 6.81 41.35
18    MA 26.92 4.69 22.04 3.03 6.89 36.65
19    MI 24.96 5.27 22.72 2.97 6.91 37.87
20    MN 22.06 3.72 14.20 3.54 8.28 29.74
21    MS 16.08 3.06 15.60 1.77 6.08 26.51
22    MO 27.56 4.04 20.98 2.55 6.82 34.39
23    MT 23.75 3.95 19.50 3.43 6.90 33.78
24    NB 23.32 3.72 16.70 2.92 7.80 31.14
25    NE 42.40 6.54 23.03 2.85 6.67 39.09
26    NJ 28.64 5.98 25.95 3.12 7.12 42.17
27    NM 21.16 2.90 14.59 2.52 5.95 25.96
28    NY 29.14 5.30 25.02 3.10 7.23 40.65
29    ND 19.96 2.89 12.12 3.62 6.99 25.62
30    OH 26.38 4.47 21.89 2.95 7.38 36.69
31    OK 23.44 2.93 19.45 2.45 7.46 32.29
32    PE 23.78 4.89 12.11 2.75 6.83 26.58
33    RI 29.18 4.99 23.68 2.84 6.35 37.86
34    SC 18.06 3.25 17.45 2.05 5.82 28.57
35    SD 20.94 3.64 14.11 3.11 8.15 29.01
36    TE 20.08 2.94 17.60 2.18 6.59 29.31
37    TX 22.57 3.21 20.74 2.69 7.02 33.66
38    UT 14.00 3.31 12.01 2.20 6.71 24.23
39    VT 25.89 4.63 21.22 3.17 6.56 35.58
40    WA 21.17 4.04 20.34 2.78 7.48 34.64
41    WI 21.25 5.14 20.55 2.34 6.73 34.76
42    WV 22.86 4.78 15.53 3.28 7.38 30.97
43    WY 28.04 3.20 15.92 2.66 5.78 27.56
44    AK 30.34 3.46 25.88 4.32 4.90 38.56
>
```

Let's now look at how to calculate some basic statistics in R. Things like mean, median, variance, minimum and maximum values are all used quite often. R makes these statistics very easy to calculate. You may suspect, mean(), median(), var(), min() and max() are the commands for the corresponding statistics. Let's take a look at these statistics in the cancer and smoking dataset.

```
> mean(smoking$LUNG)
[1] 19.65318
> median(smoking$LUNG)
[1] 20.32
> var(smoking$LUNG)
[1] 17.87701
> min(smoking$LUNG)
[1] 12.01
> max(smoking$LUNG)
[1] 27.27
>
```

There are a few commands in R that are called general commands. This is because depending on how you call the command you will have different things returned by R. One of these commands is summary(). Let's see how it works

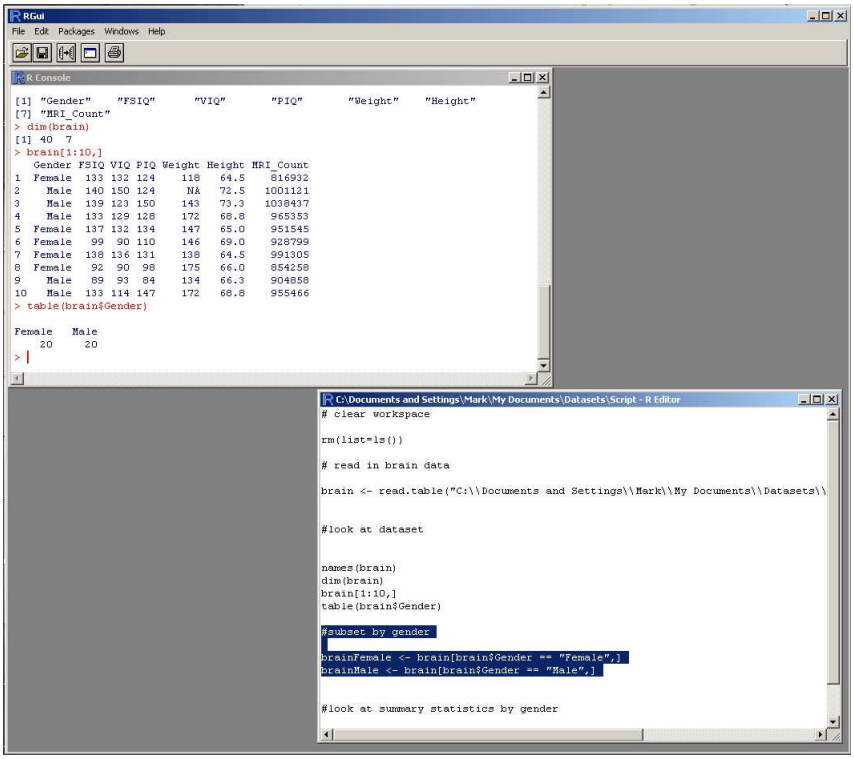
when applied to our cancer and smoking dataset.

```
> summary(smoking)
    STATE                CIG             BLAD            LUNG
 Length:44          Min.   :14.00   Min.   :2.860   Min.   :12.01
 Class :character   1st Qu.:21.23   1st Qu.:3.208   1st Qu.:16.44
 Mode  :character   Median :23.77   Median :4.065   Median :20.32
                    Mean   :24.91   Mean   :4.121   Mean   :19.65
                    3rd Qu.:28.10   3rd Qu.:4.782   3rd Qu.:22.81
                    Max.   :42.40   Max.   :6.540   Max.   :27.27
      KID              LEUK
 Min.   :1.590   Min.   :4.900
 1st Qu.:2.458   1st Qu.:6.532
 Median :2.845   Median :6.860
 Mean   :2.795   Mean   :6.830
 3rd Qu.:3.112   3rd Qu.:7.207
 Max.   :4.320   Max.   :8.280
>
```

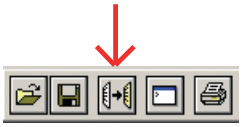
As you can see, the summary command in this instance returns the minimum, maximum, mean and all three quartiles for each variable in the dataset. The STATE field is summarized as well, but since the variable is not numeric the same statistics are not returned.

The last topic we will discuss in this lesson is using scripts. Thus far we have been going command by command. Most times it is beneficial to create a script to run your analysis in R. The main benefit is that you have a record of your analysis and it makes it much easier to replicate your work. Additionally, if you commonly use a single procedure to analyze your work, by using a script you can make small changes to the script depending on your dataset and save a lot of time.

To start a new script, go to the File drop down menu and select “New script”. This opens up a new window that you can type your commands into.




After you have entered in the commands you want to run, you can either run select lines in the script or the whole script file. To run a line of script, highlight the line with your mouse and select the middle icon on the tool bar at the top of the screen. Make sure that you do not have anything typed into the current command line when you are running scripts.



Let's look at an example of running scripts.

```
# clear workspace

rm(list=ls())

# read in brain data

brain <- read.table("C:\\Documents and Settings\\Admin\\Desktop
\\brain_size_data.txt", header=T, sep="\t")

# look at dataset
```

```
names(brain)
dim(brain)
brain[1:10,]
table(brain$Gender)

# subset by gender

brainFemale <- brain[brain$Gender=="Female",]
brainMale <- brain[brain$Gender=="Male",]

#look at summary statistics by gender

summary(brainFemale)
summary(brainMale)
```

As you can see, this script contains commands to load a new dataset. It looks at some of the characteristics of the dataset, subsets the dataset and looks at some summary statistics. Also, you may notice the lines starting with the pounds sign. The # tells R to ignore all text to the right of the sign. This allows you to make comments in your script so you can remember what you were trying to do. This also makes it much easier to troubleshoot problems later on. Remember that if you are using different computers with your scripts you may have to change the directory that your dataset is stored within your script. Now let's run this script and see what happens.

```
> # clear data
>
> rm(list=ls())
>
> # read in brain data
>
> brain <- read.table("C:\\Documents and Settings\\Admin\\Desktop
\\brain_size_data.txt", header=T, sep="\t")
>
> # look at dataset
>
> names(brain)
[1] "Gender"    "FSIQ"      "VIQ"       "PIQ"       "Weight"    "Height"
[7] "MRI_Count"
> dim(brain)
[1] 40  7
> brain[1:10,]
   Gender FSIQ VIQ PIQ Weight Height MRI_Count
1  Female  133 132 124    118   64.5    816932
2    Male  140 150 124     NA   72.5   1001121
3    Male  139 123 150    143   73.3   1038437
4    Male  133 129 128    172   68.8    965353
5  Female  137 132 134    147   65.0    951545
6  Female   99  90 110    146   69.0    928799
7  Female  138 136 131    138   64.5    991305
```

```
8  Female   92  90  98    175   66.0    854258
9    Male   89  93  84    134   66.3    904858
10   Male  133 114 147    172   68.8    955466
> table(brain$Gender)

Female   Male
    20     20
>
> # subset by gender
>
> brainFemale <- brain[brain$Gender=="Female",]
> brainMale <- brain[brain$Gender=="Male",]
>
> #look at summary statistics by gender
>
> summary(brainFemale)
    Gender               FSIQ             VIQ             PIQ
 Length:20          Min.   : 77.00   Min.   : 71.0   Min.   : 72.0
 Class :character   1st Qu.: 90.25   1st Qu.: 90.0   1st Qu.: 93.0
 Mode  :character   Median :115.50   Median :116.0   Median :115.0
                    Mean   :111.90   Mean   :109.5   Mean   :110.5
                    3rd Qu.:133.00   3rd Qu.:129.0   3rd Qu.:128.8
                    Max.   :140.00   Max.   :136.0   Max.   :147.0
     Weight          Height        MRI_Count
 Min.   :106.0   Min.   :62.00   Min.   :790619
 1st Qu.:125.8   1st Qu.:64.50   1st Qu.:828062
 Median :138.5   Median :66.00   Median :855365
 Mean   :137.2   Mean   :65.77   Mean   :862655
 3rd Qu.:146.2   3rd Qu.:66.88   3rd Qu.:882669
 Max.   :175.0   Max.   :70.50   Max.   :991305
> summary(brainMale)
    Gender               FSIQ             VIQ              PIQ
 Length:20          Min.   : 80.00   Min.   : 77.00   Min.   : 74.0
 Class :character   1st Qu.: 89.75   1st Qu.: 95.25   1st Qu.: 86.0
 Mode  :character   Median :118.00   Median :110.50   Median :117.0
                    Mean   :115.00   Mean   :115.25   Mean   :111.6
                    3rd Qu.:139.25   3rd Qu.:145.00   3rd Qu.:128.0
                    Max.   :144.00   Max.   :150.00   Max.   :150.0

     Weight          Height        MRI_Count
 Min.   :132.0   Min.   :66.30   Min.   : 879987
 1st Qu.:148.8   1st Qu.:68.90   1st Qu.: 919529
 Median :172.0   Median :70.50   Median : 947242
 Mean   :166.4   Mean   :71.43   Mean   : 954855
 3rd Qu.:180.8   3rd Qu.:73.75   3rd Qu.: 973496
 Max.   :192.0   Max.   :77.00   Max.   :1079549
 NA's   :  2.0   NA's   : 1.00
>
```

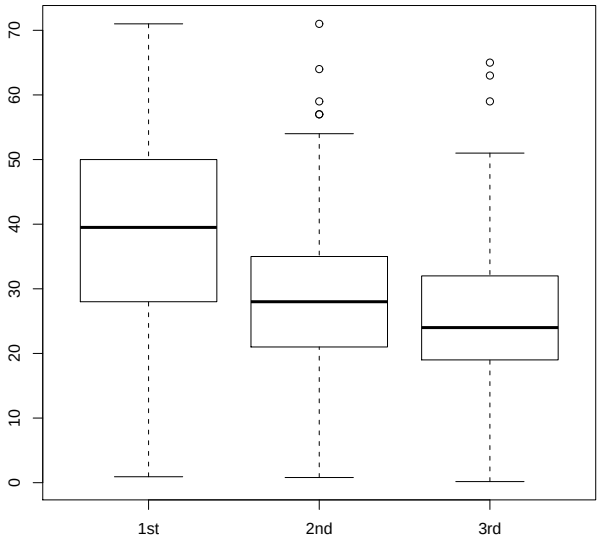
As you can see each command was run, just as before when we ran these commands piece wise. This technique may seem like overkill now just looking at these simple analyses, but as your statistical analysis becomes more sophisticated, using scripts is a must.

In this lesson we have demonstrated how easy it is to manipulate data in R for analysis and how to use scripts to save work for further use. The next lesson will demonstrate one of R's most popular features, plotting and graphics.

# 4 Plotting

Now let's take a look at how to create plots. This is another of R's most beneficial features. Just like the other features that we have looked at thus far, we will create plots in R using command line. Let's first take a look at our Titanic dataset that we used previously.

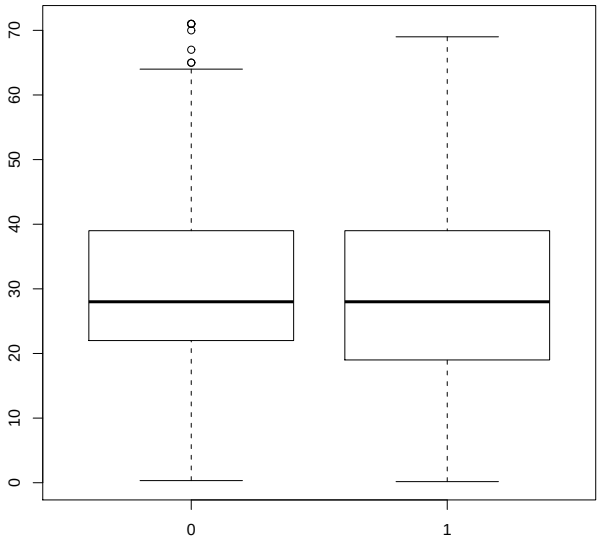
```
> titanic <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
titanic.txt", header=T, sep="\t")
> names(titanic)
[1] "Name"     "PClass"   "Age"      "Sex"      "Survived"
> boxplot(Age ~ PClass, data=titanic)
>
```




Let's see what has been plotted for us. We can see that there were three classes on board the titanic, and the boxplot function has plotted boxplots of the age of the passengers on the Titanic and separated the passengers out by class. Also if you look at the command, you will notice that we also had to tell R where to find the Age and PClass data.

Let's take a look at another plot with this dataset. How about looking at age again but separating the group by whether the passenger survived or not.

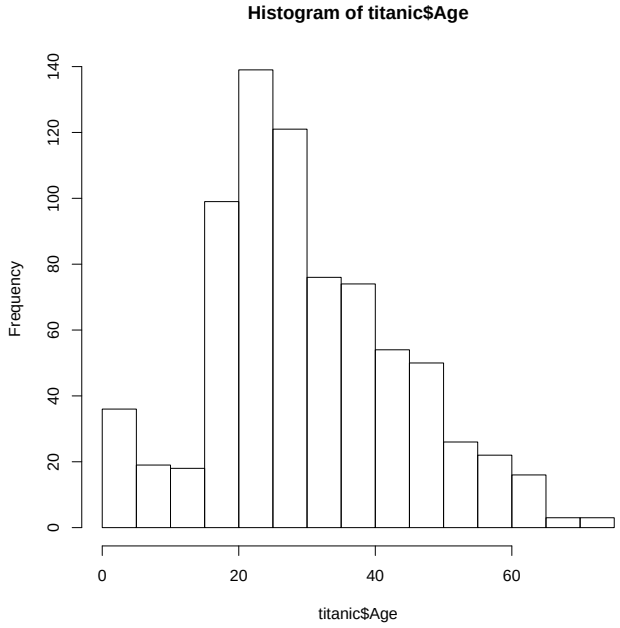
```
> boxplot(Age ~ Survived, data=titanic)
>
```




One thing you may have noticed in these examples is that the variable that we plotted, age, is continuous where as the variable that we separated by is discrete. If this constraint is not met, you will receive an error message from R.

Let's now take a look at another great diagnostic plot. The histogram is very useful to get an idea of what a variable looks like. Let's see what the distribution of the age of the Titanic passengers looked like.

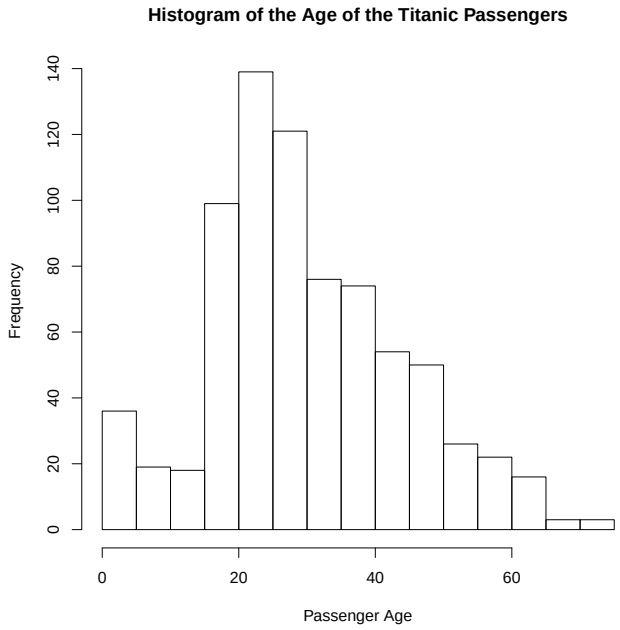
```
> hist(titanic$Age)
>
```




You can see from this plot that the majority of the passengers on the Titanic were in between 20 and 30 years old.

Now what if we wanted to include this plot in a report or homework assignment? While there are default titles and labels on the axes, I don't think we would want to turn in something that says the title of this plot is Histogram of `titanic$Age` . In R you have control over the titles, the axes labels and just about anything else you want to change or customize. Let's add some better labels to this plot.

```
> hist(titanic$Age, main = "Histogram of the Age of the Titanic Passengers",
  xlab  = "Passenger Age")
>
```



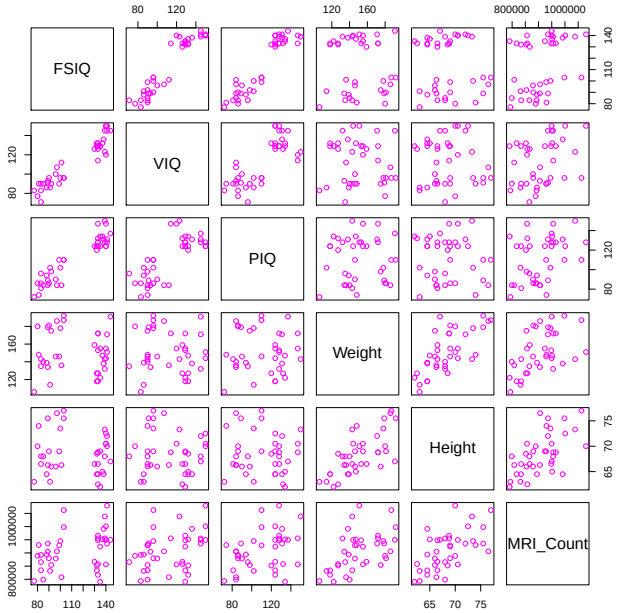

Now, doesn't that look better?

Let's now take a look at some advanced plotting techniques. We'll start by reloading our Brain Size dataset.

```
> brain <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
brain_size_data.txt", header = T, sep="\t")
> brain[1:10,]
   Gender FSIQ VIQ PIQ Weight Height MRI_Count
1  Female  133 132 124    118   64.5    816932
2    Male  140 150 124     NA   72.5   1001121
3    Male  139 123 150    143   73.3   1038437
4    Male  133 129 128    172   68.8    965353
5  Female  137 132 134    147   65.0    951545
6  Female   99  90 110    146   69.0    928799
7  Female  138 136 131    138   64.5    991305
8  Female   92  90  98    175   66.0    854258
9    Male   89  93  84    134   66.3    904858
10   Male  133 114 147    172   68.8    955466
>
```

Recall that this dataset has six continuous variables along with the gender of each observation. Let's start by graphing all six of our continuous variables.

```
> plot(brain[,2:7], col = 6)
>
```

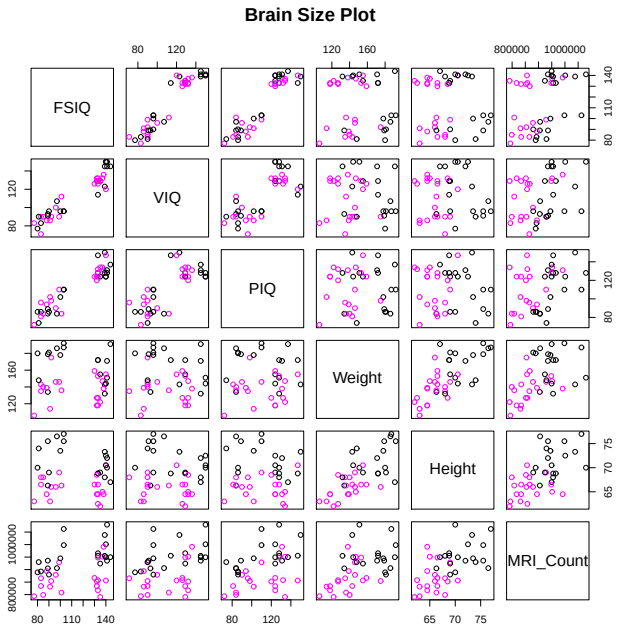



The plot() function is another general function in R just like summary was in the previous section. Depending on the type of object we provide and the type of data in the dataset, plot() will return different plots. You can see from the command entered, we asked R to plot all rows in the brain dataset but only columns two through 7. The second part of the plot command, col = 6, told R that I would like to change the color to 6 which corresponds to a fuscia. This is very useful in looking at the relationships between your data's variables, but let's take this a step further. What if we want to determine whether or not each of these data points in the graph are male or female. We could mark the female observations red if we want. The easiest way to do this is to create another column in the brain size dataset to denote the color based on the gender of the observation.

```
> brain$GenderInt <- 5*(brain$Gender=="Female") + 1
>
```

We have now created a variable that will be 6 if the observation's gender is female and 1 if the observation's gender is male. Now let's create our graph again with this color distinction and add a title.

```
> plot(brain[,2:7], col = brain$GenderInt, main = "Brain Size Plot")
>
```

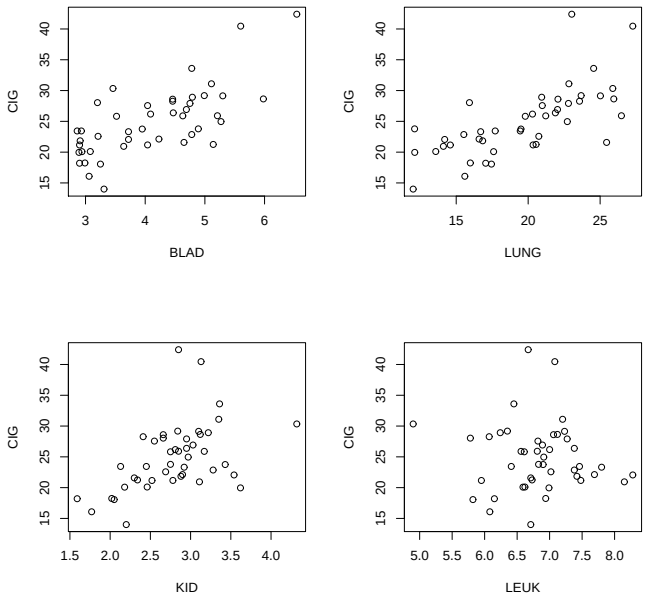



Let's take a look at another useful feature in R. Say we want to produce multiple plots on one screen. R makes this very easy to do. Let's go back to our Smoking and Cancer dataset to look at an example.

```
> smoking <- read.table("C:\\Documents and Settings\\Admin\\Desktop\\
cancer_and_smoking.txt", header = T, sep="\t")
> smoking[1:10,]
>
```

Let's say that we want to create four plots displaying smoking versus bladder, lung, kidney and leukemia cancers.

```
> par(mfrow = c(2,2))
> plot(CIG~BLAD, data = smoking)
> plot(CIG~LUNG, data = smoking)
> plot(CIG~KIN, data = smoking)
> plot(CIG~LEUK, data = smoking)
>
```



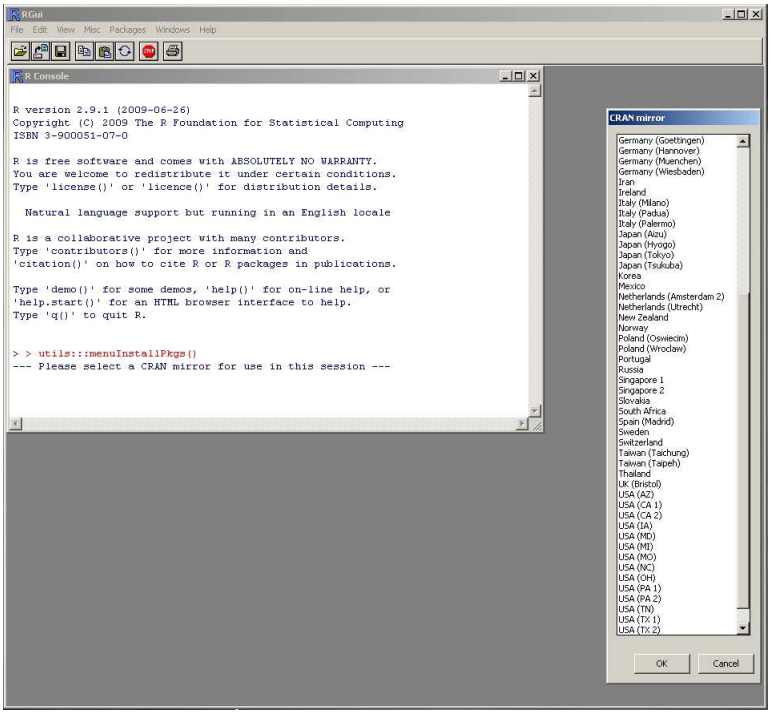

You can see that we have now created a series of plots arranged in a 2 by 2 array. Depending on how you would like the plots arranged, you could have asked R to set them up in a 4 by 1 array or 1 by 4 just by changing the initial dimensions in the mfrow command.

This just scratches the surface of the graphical capabilities of R; however, it should allow you to create most of the plots you will need to use during a beginning statistics course. Remember, if you ever forget what plotting options you have, it's easy to look up in the help() command.

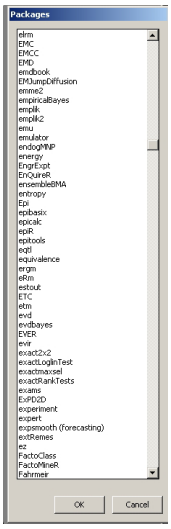
# 5 Installing Packages

The base R package that you have installed contains the functionality for the majority of the analysis you execute in a beginning statistics course. There are numerous external packages that you can add to augment the functions in the base package. Let's go through the process of adding an R package.

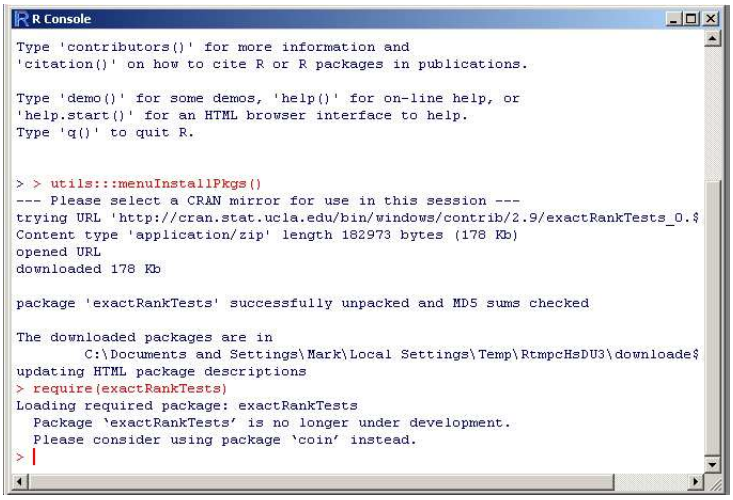
First, select the "Package" menu and select "Install package(s)". Then select your desired CRAN mirror in the list that appears; the ones in the United States are towards the bottom of the list.




In this example we are going to install the "exactRankTests" package. After selecting the desired package, R will download the required files and install them on your computer.




Now, in order to use the features in the package you have just installed, you must tell R to load the package. Loading the package must be done each R session. This is done by using the require() or library() command with the exact name of the package. Now you are ready to use the functions in your package. Full documentation for all packages can be found on the R CRAN websites.




This concludes this introduction to R. In these lessons we have learned the basic R skills that you will need for a beginning statistics course. If you have specific questions on other features that were not covered in this tutorial, R is very well documented with help files. Also online manuals that can be found at http://www.r-project.org under the document and manuals menu.