

Introduction to R

Jennifer Moore

R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.

RStudio (2012). RStudio: Integrated development environment for R (Version 0.96.122) [Computer software]. Boston, MA. Retrieved September 27, 2014. Available from <http://www.rstudio.org/>

Table of Contents

Installing R.....	3
Getting Started	4
Help in R.....	5
Links to R Manuals/Basic Lessons.....	5
Setting the Working Directory	6
Saving your Work.....	6
R Packages & Libraries	7
Entering Data into R.....	8
Comments in R	10
Data Types and Structure	11
Descriptive Statistics	16
Missing Values	18
Plotting.....	18
Writing Functions.....	26
Data Analysis – Simple Tests	27

Installing R

Download R using one of the following links – depending on whether you have a PC or a Mac. After you have downloaded R, you can download R Studio. R Studio is an open source integrated development environment (IDE) for R. Both R and R studio are free.

Windows Users:

<http://cran.at.r-project.org/bin/windows/base/>

Mac Users:

<http://cran.r-project.org/bin/macosx/>

R-Studio:

<http://www.rstudio.com/products/rstudio/download/>

Getting Started

In R Studio, there are 4 main windows.

- (1) Console Window
- (2) Environment/History Window
- (3) Files/Plots/Packages/Help Window
- (4) Script Window (does not open automatically)

The **Console Window** is where the code is run. You can type R commands directly into this window, or you can run R code or script (which is just a collection of valid R commands) from a script file. All of your commands and outputs are also displayed in the console.

The **Environment/History Window** is in the top right of the screen. The Environment tab lists all of the data and values that have been created and saved. This allows you to see what each variable you have created represents. You can also import data into R directly using this window. The History tab keeps track of all of the code that you have run. You can save this into a file.

The **Files/Plots/Packages/Help Window** is in the bottom right of the screen. The Files tab shows all of the files on your computer; you can also set working directory (see below for details), create new folders, and copy or move files from the Files tab. You can open data files directly from this window to look at, but they must be imported into R Studio for you to be able to use or analyze the data. The Plot tab is where all of the plots you create are displayed, and where they can be saved into files. The Packages tab is a list of all currently installed packages, the place to update packages, and the place to install new packages. The Help tab displays the help files for all of the commands and packages.

The **Script Window** is created by clicking File > New File > New Script. An R script can be saved, so you can reopen it to continue working on a project, and it can be sent to another person for them to open and edit.

It is best to organize all code within an R Script.
Press <CTRL Enter> to run code from script in the Console Window or use the 'Run' button at the top of the Script Window.

Help in R

Introduction to R manual

<http://cran.r-project.org/doc/manuals/R-intro.pdf>

R help mailing list

<http://r.789695.n4.nabble.com/R-help-f789696.html>

R built in help

The help files can be searched in the Help tab in the bottom right corner. In addition, the following commands can be entered into the Console Window to open the specified files (in this case help on the plot function)

```
> help(plot)
> ?plot
> ??plot
```

The single question mark `?` is used to search for a particular function, the double question mark `??` is used if you do not know the function you want to use, but want to search for potential options.

R built in examples

The example files can be accessed by entering the following command into the Console Window. This will bring up examples of how the function can be used.

```
> example(plot)
```

You can also find R help by searching Google!

Links to R Manuals/Basic Lessons

Here are a few websites with tutorial on learning the basics in R. There are a lot of tutorials, manuals and presentations about various aspects of R available online, so if you aren't interested in these, there are plenty more out there!

<http://tryr.codeschool.com/levels/1/challenges/1>

<https://www.datacamp.com/courses/introduction-to-r>

<http://swirlstats.com/students.html>

Setting the Working Directory

Before beginning work in R/R studio it is important to set the working directory. This is the folder where all files that you are going to access are located, as well as where all files that you create will be saved.

You can set working directory in three ways: using Files tab and using R commands. In the Files tab in the bottom left corner, locate the folder that you want as your working directory. Once you have clicked on the folder and it is open, under 'More' (menu at the top of the Files tab) click 'Set as Working Directory'. An alternative way of achieving the same goal is to use R Studio main menu -> Session -> Set working directory.

You can also set working directory using the `setwd` (stands for **set** working **d**irectory) command as follows (you need know the full name of the directory or folder):

```
> setwd("C:/Statistics/R/")  
> setwd("C:\\Statistics\\R\\")
```

Note: All slashes needs to be forwards slashes (/) or double backward slashes (\\). Also, R is case sensitive, so make sure capital letters are used only when needed.

You can also find out what your current working directory is by using the `getwd` function:

```
> getwd()
```

Saving your Work

You can save everything in all of the windows in R Studio (data, R objects etc.) using `save.image` command.

```
> save.image("File.R")
```

This will create a file called *File.R* in your working directory.

```
> load("File.R")
```

This will load your data back so you can keep working where you left off.

This is helpful where you are working with code that takes a long time to run because you do not have to rerun it every time you want to view your results.

You can also save just the R script or single plots that you have created.

R Packages & Libraries

The functions in R are organized in packages or libraries. The commonly used packages are preinstalled within R. However, if you need to use more specialized packages, you will need to install and then load the package before you can use functions available in that package.

Packages are installed and loaded in the Packages tab in the bottom right corner or through the dropdown menu at the top of the screen (Tools -> Install Packages).

If a package is already installed on your computer, it will be listed in this tab. To load this package so you can use it, hit the checkmark next to the name of the package.

If a package is not installed, click the 'Install' button at the top of the tab, type in the name of the package, and click 'Install'. Once it has been installed click the checkmark next to the name to load it.

Installed packages can also be loaded from the command line or within a script using the library function.

```
>library(ggplot2)
```

This would load the previously installed package 'ggplot2'.

A list of all available packages can be found here:

http://cran.r-project.org/web/packages/available_packages_by_name.html

Entering Data into R

Data can be entered into R using a variety of methods.

- (1) Manually – entered into the command line
- (2) Imported – entered in Excel (or other software – see format below), saved as .csv or .txt files, and imported into R
 - a. Using the Environment tab
 - b. Using 'read' commands

Manual Entry

Data can be typed directly into an R script.

As a calculator

```
> 10 + 5  
[1] 15
```

Compute a sum

```
> sum(1,5,10)  
[1] 16
```

Create two vectors of numbers called Numbers and Numbers2

```
> Numbers <- c(5,4,7)  
> Numbers2 = c(5,4,7)  
> Numbers  
[1] 5 4 7  
> Numbers2  
[1] 5 4 7  
> print(Numbers)  
[1] 5 4 7
```

Notes

- The <- and = symbols are interchangeable in R. The two vectors Numbers and Numbers2 are identical.
- Numbers is the name of the vector containing the numbers 5, 4, 7. To display the contents of the vector type 'Numbers' into the console, or type print(Numbers)
- The c used in the vector above stands for 'concatenate' or 'combine'. It is used whenever a list of data points is entered into R.

Importing CSV and TXT Files

Data files can only be imported into R if they are saved as a CSV or TXT file. You can enter data in Excel or other spreadsheet programs, and then save the file into one of these two formats before it can be imported for use in R.

A CSV file is recommended for use in R.

Using the Environment tab

- In the Environment tab, in the top right corner click on 'Import Dataset', and then 'From Text File...'.
• Choose the file that you want to import.
• Choose a name for the data – enter into the 'Name' blank (This name cannot contain any spaces).
• Choose yes or no depending on whether or not the file has a Heading (column names).
• Choose the separator, decimal, quote, and na.strings depending on your data.
• Click import.

Read Commands

The following command can be typed into the console to import the data.

For a CSV file:

```
> data <- read.csv("data.csv", header = TRUE)
```

For a text file:

```
> data <- read.table("data.txt", header = TRUE, sep=" ")
```

The first argument is the file name (found within the working directory set earlier).

The second argument, `header`, refers to whether or not the column names are listed in the first line of the file, `TRUE` is used for yes, `FALSE` for no.

The third argument, `sep`, refers to what is separating the values. A CSV uses commas, but a text document could use semicolons, colons, etc.

Import the beaver dataset into R using either of the two methods explained above. Make sure to save it as a csv file in Excel before importing it into R. When you import it into R, name the variable containing the data 'beaver'. We will use it later! You can view the data after it is imported in R by using the command `View`, or by double clicking the file in the Environment tab.

Comments in R

Comments are notes that you include in your code, but that are not run. R does not run any commands included in your comments.

It is good practice to always put comments into your code, so you or anyone else who access your code later knows what you were doing. This might seem unnecessary for simple code, but will be very useful with more complicated code.

In R, you can comment a line by putting the # sign in front of the text. If you want to comment a large chunk of code all at once, highlight the code and then use the dropdown menu (Code -> Comment/Uncomment Lines).

```
>#this is a comment
```

```
>a = c(1,2,3)
```

```
>#creates a vector, a, containing the numbers 1, 2, and 3
```

```
>sum(a)
```

```
>#calculates the sum of the numbers in vector a
```

Data Types and Structure

R can use many data types, such as numeric, categorical, and ordinal. They are organized in various structures, which dictates how they are manipulated, accessed and used by R.

Each data point or a variable can be numeric, a character, or a factor.

- Numeric – a number (e.g. 1, 5.3, -2, 4)
- Character – string of text (e.g. red, one, monkey)
- Factor – level (e.g. male/female, high/low)

Data in R can be organized as

- (1) vectors
- (2) matrices
- (3) data frames

Vectors

Vectors are one-dimensional arrays containing numbers, or characters.

```
>a <- c(1,2,5.3,6,-2,4) # numeric vector
>b <- c("one","two","three") # character vector
>c <-c(1, "one") #mixed vector
```

Vectors can also be created using sequences of numbers, or repetitions of the same number.

```
>5:9
[1] 5 6 7 8 9
>seq(5,9)
[1] 5 6 7 8 9
>seq(5,9,0.5)
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0
>rep(5,3)
[1] 5 5 5
```

Note: Characters must be entered inside double quotes.

Access the elements of the vectors using their position in the vector in brackets []

```
> a[3] #3rd element of vector a
[1] 5.3
>a[c(2,4)] # 2nd and 4th elements of vector a
[1] 2 6
>a[3:5] #return the 3rd, 4th, and 5th element of vector a
```

```
[1] 5.3 6.0 -2.0
```

Add elements to the vector

```
> a[7] <- 0 #add a 7th element to the vector
```

```
> a
```

```
[1] 1.0 2.0 5.3 6.0 -2.0 4.0 0.0
```

Change elements in the vector

```
> a[1] <- 0 #change the first element to 0 instead of 1
```

```
> a
```

```
[1] 0.0 2.0 5.3 6.0 -2.0 4.0 0.0
```

Add a value to every element of the vector

```
> a + 1
```

```
[1] 1.0 3.0 6.3 7.0 -1.0 5.0 1.0
```

Multiply a value to every element of the vector (you can also subtract or divide, etc.)

```
> a * 2
```

```
[1] 0.0 4.0 10.6 12.0 -4.0 8.0 0.0
```

Add two vectors together (or subtract, multiple, divide)

```
> b <- seq(1,7)
```

```
> a+b
```

```
[1] 1.0 4.0 8.3 10.0 3.0 10.0 7.0
```

The above commands simply output the desired results – if you want to use the new vector it must be saved as a new variable.

```
> c <- a+b
```

```
[1] 1.0 4.0 8.3 10.0 3.0 10.0 7.0
```

```
#the sum of vectors a and b is now saved as variable c
```

Matrices

Matrices are two-dimensional, for example a 4 x 5 matrix has 4 rows and 5 columns. All values within the matrix must be of the same type; i.e. all numeric or all characters

```
> mymatrix <- matrix(1:4, nrow=2, ncol=2)
```

```
> mymatrix
```

```
  [1] [2]
```

```
[1,] 1 3
```

```
[2,] 2 4
```

The values of a matrix are accessed using their position, like for vectors, but for matrices it is necessary to designate the row and column.

```
> mymatrix[1,2] #the value in row 1 column 2
[1] 3
> mymatrix[2,1] #the value in row 2 column 1
[1] 2
> mymatrix[,2] #the values for all rows, column 2
[1] 3 4
> mymatrix[1,] # the values for row 1, and all columns
[1] 1 3
> mymatrix[1:2,] #the values from row 1 to 2, and all columns
  [,1] [,2]
[1,]  1  3
[2,]  2  4
```

Change the value in row 1 column 2 to 0

```
> mymatrix[1,2] <- 0
> mymatrix[1,2]
[1] 0
```

Data Frames

Data frames are a more general form of a matrix. The values do not have to be of the same type, but instead one data frame can contain numbers, characters, and factors.

```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", "blue")
> f <- factor(c("high", "low", "high", "high"))
> mydata <- data.frame(d,e,f)
> names(mydata) <- c("ID", "Color", "Level")
> mydata
  ID Color Level
1  1  red  high
2  2 white  low
3  3  red  high
4  4 blue  high
```

The values of a data frame can be accessed in a variety of ways.

```
> mydata[1] #calls column 1
  ID
1  1
2  2
```

```
3 3
4 4
```

```
> mydata["ID"] #another way to access column 1 – using the column name
ID
1 1
2 2
3 3
4 4
```

```
> mydata$ID #last way to access column 1 – also using the column name
[1] 1 2 3 4
```

```
> mydata[1,] #access the first row
ID Color Level
1 1 red high
```

Useful Functions for Data Types

Note: R has many datasets built into the program that you can use for practicing. Type `data()` into the console to see a list of all of the available datasets. We will use the iris dataset. If you type `help(iris)` you can read all about the data included in this dataset.

```
> str(iris) # structure of the dataframe
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> head(iris) #displays first 10 rows of data
> tail(iris) #displays last 10 rows of data
```

```
> nrow(iris) #number of rows
[1] 150
> ncol(iris) #number of columns
[1] 5
> length(iris$Sepal.Length) #number of data points in the column
[1] 150
```

```
> class(iris) #class or type of object
[1] "data.frame"
```

```
> names(mydata) # variable names
```

```
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

The format of data can also be changed after it is entered. For example, a vector of characters or numbers could be changed to factors, or a matrix could be changed into a dataframe.

```
> str(d)
num [1:4] 1 2 3 4
> d <- factor(d)
> str(d)
Factor w/ 4 levels "1","2","3","4": 1 2 3 4
```

Vector 'd' was changed from a vector of numbers of a vector of factors.

```
> str(mymatrix)
int [1:2, 1:2] 1 2 3 4
> mymatrix <- as.data.frame(mymatrix)
> str(mymatrix)
'data.frame': 2 obs. of 2 variables:
 $ V1: int 1 2
 $ V2: int 3 4
```

The matrix 'mymatrix' was changed to a dataframe.

Subsetting Data

Sometimes if we have a large dataset for multiple categories, we might want to subset it into smaller datasets for analysis.

For example, let's look at the iris dataset, which gives us information on three different species of iris.

We can create separate datasets for each of the species.

```
> setosa <- subset(iris, iris$Species == "setosa")
#this creates a new dataframe called 'setosa' which contains all columns of data for
just the species setosa
> versicolor <- subset(iris, iris$Species == "versicolor")
#this creates a new dataframe called 'versicolor' which contains all columns of data
for just the species versicolor
```

We can also subset based on criteria – in this case such as petal length or width.

```
> iris2 <- subset(iris, iris$Petal.Length < 2)
#this dataframe only contains the flowers that have a petal length less than 2
```

Descriptive Statistics

We will use the beaver dataset that we imported earlier to explore basic descriptive statistics. But first, we need to look at the structure of the dataset.

```
> str(beaver)
'data.frame': 51 obs. of 2 variables:
 $ Year   : int  1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 ...
 $ Population: int 65276 71699 59199 123045 70269 102607 87759 117276 132897
161276 ...
```

This dataset has two variables, Year and Population. Let's explore the 'Population' variable.

```
> beaver$Population #displays the data in the Population column
```

Minimum – smallest value

```
> min(beaver$Population)
[1] 59199
```

Maximum – largest value

```
> max(beaver$Population)
[1] 1071145
```

Range (Minimum and Maximum)

```
> range(beaver$Population)
[1] 59199 1071145
```

Median – middle number when all of the values are organized from smallest to largest

```
> median(beaver$Population)
[1] 455595
```

Quantile

```
> quantile(beaver$Population)
 0%  25%  50%  75% 100%
59199 199130 455595 588137 1071145
```

Mean is a measure of central tendency of the data, and is calculated as:

$$\bar{X} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum x_i}{n}$$

```
> mean(beaver$Population)
[1] 410710.2
```

Variance is a measure of spread of the data, and is calculated as

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{X})^2$$

```
> var(beaver$Population)
[1] 54920644842
```

Standard deviation is just the square root of variance (how much the values typically vary from the average value):

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{X})^2}$$

```
> sd(beaver$Population)
[1] 234351.5
```

Standard Error – the variability in which a sample estimates a population (how much the values typically vary around the sample mean of a population)

$$se = \frac{s}{\sqrt{n}}$$

There is no built-in function for standard error, so we will use a combination of other functions.

```
> sd(beaver$Population)/sqrt(length(beaver$Population))
[1] 32815.78
```

Standard deviation and standard error are different measures. Standard deviation quantifies scatter – how much the values vary. Standard error quantifies precision – how close your values match the true values. Standard error takes into account standard deviation and sample size.

The summary command calculates a group of the descriptive statistics all at once.

```
> summary(beaver$Population)
  Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
59200 199100 455600 410700 588100 1071000
```

All of the values are rounded to the nearest integer before these values are calculated, which is why they are slightly different than the values calculated above.

Missing Values

Datasets are rarely perfect, and there is often data missing. There are a couple of ways in R to deal with missing data.

```
> a <- c(1,2,3,NA,5)
> sum(a)
[1] NA
> sum(a, na.rm = TRUE)
[1] 11
```

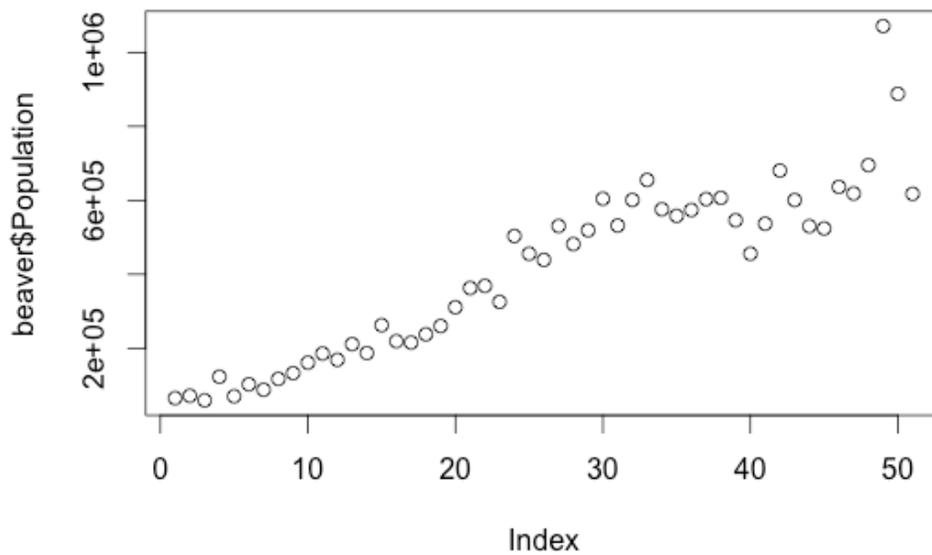
Plotting

Basic Plot

We will plot the beaver population numbers.

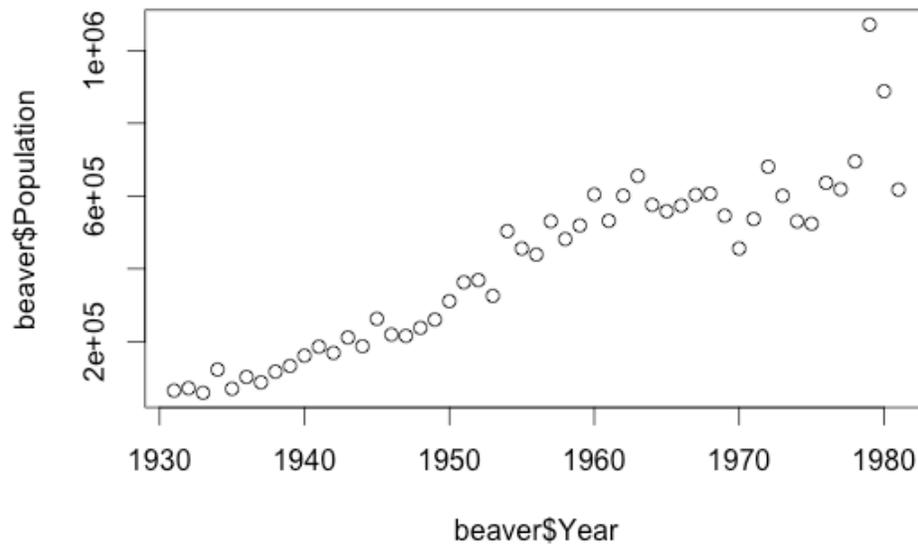
```
> plot(beaver$Population)
```

Creates a plot with an index on the x-axis and beaver populations on y-axis



```
>plot(beaver$Year, beaver$Population)
```

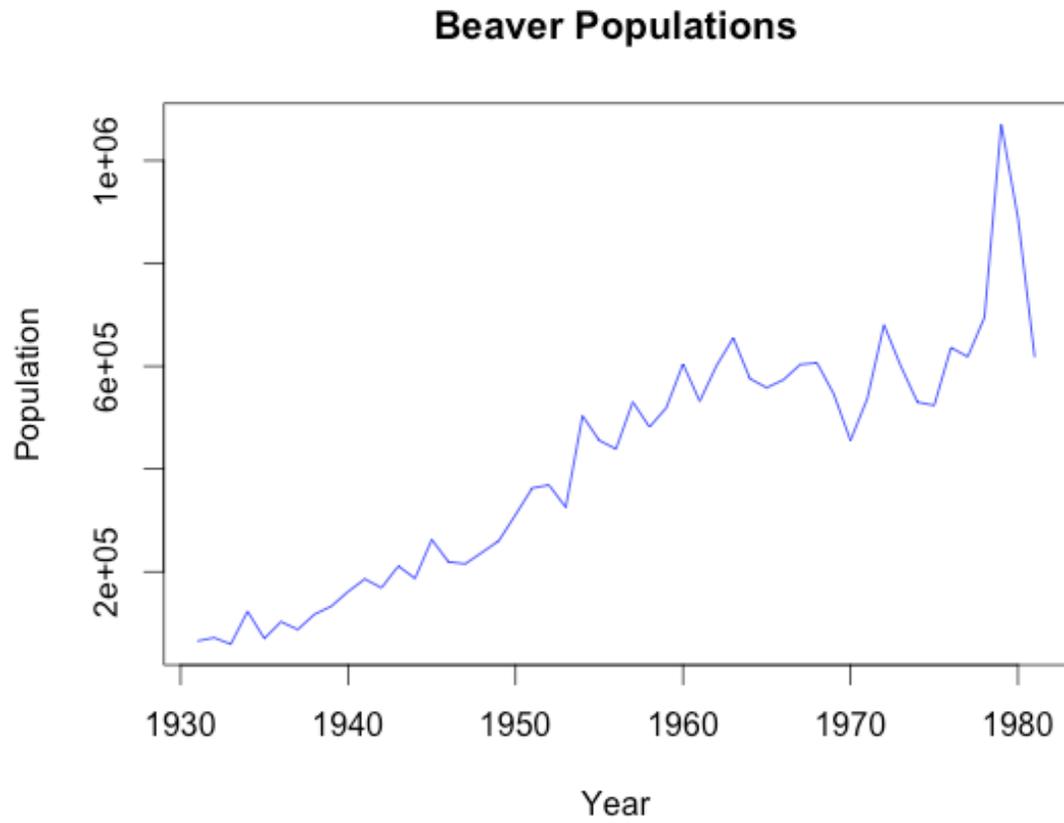
Creates a plot with years on the x-axis and beaver populations on the y-axis



There are many more arguments that can be added to the plot command to customize the graph.

- type – the type of plot to be drawn
 - Points (default)
 - Lines
- main – the title of the plot
- xlab, ylab – the axis labels
- xlim, ylim – the range of values on each axis
- col – color of symbols

```
> plot(beaver$Year, beaver$Population, type = "l", main = "Beaver Populations", xlab = "Year", ylab = "Population", col = "blue")
```

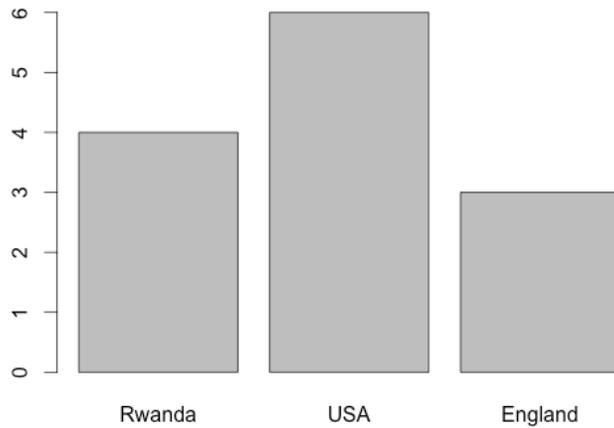


Other Common Plots

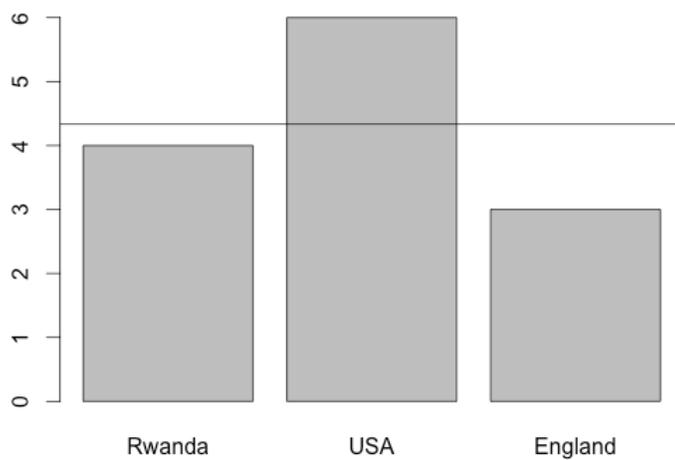
Lets use the iris data we looked at earlier to look at some other common plots.

Create a barplot for the following vector and add labels for each bar.

```
> data <- c(4,6,3)
> barplot(data)
> names(data) <- c("Rwanda", "USA", "England")
> barplot(data)
```

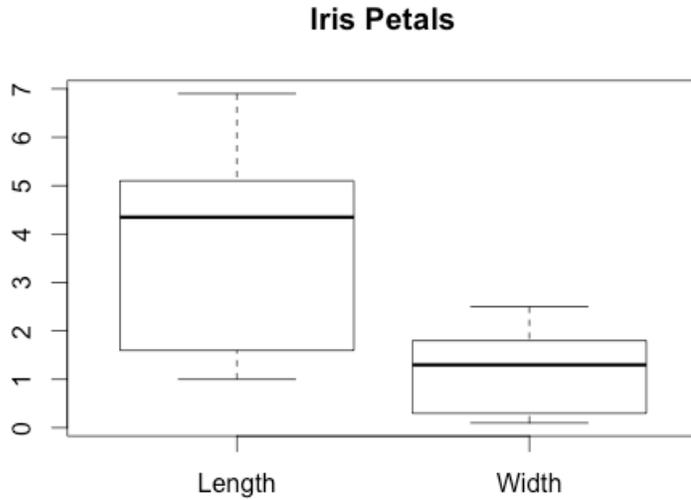


```
>abline(h=mean(data)) #adds a horizontal line to the graph
```



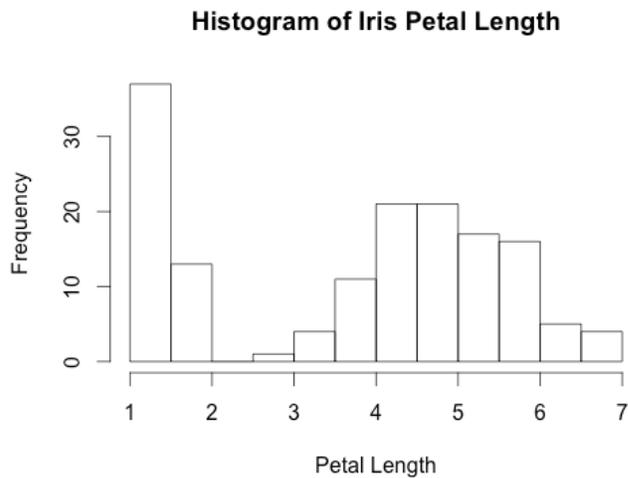
Create a side-by-side boxplot for two variables in the iris dataset.

```
> boxplot(iris$Petal.Length, iris$Petal.Width, main = "Iris Petals", names = c("Length", "Width"))
```

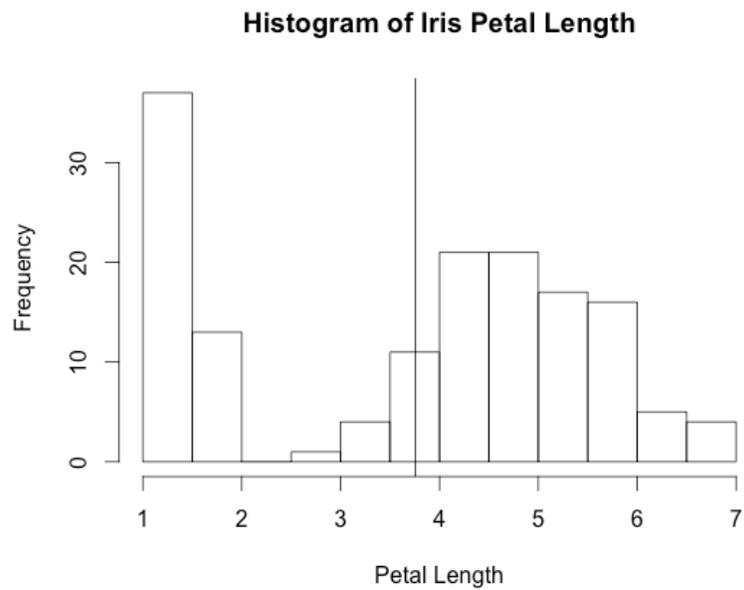


Create a histogram of the petal length from the iris dataset.

```
> hist(iris$Petal.Length, xlab = "Petal Length", main = "Histogram of Iris Petal Length")
```



```
>abline(v=mean(iris$Petal.Length))
```

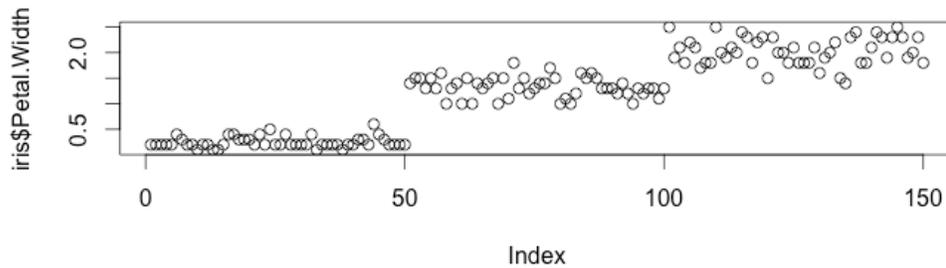
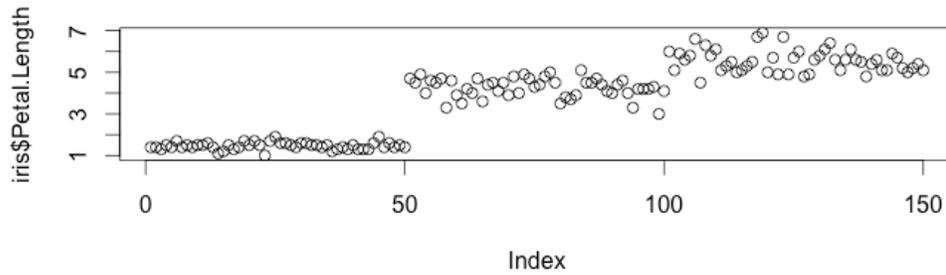


Multiple Plots

It is possible to put multiple plots on the same page.

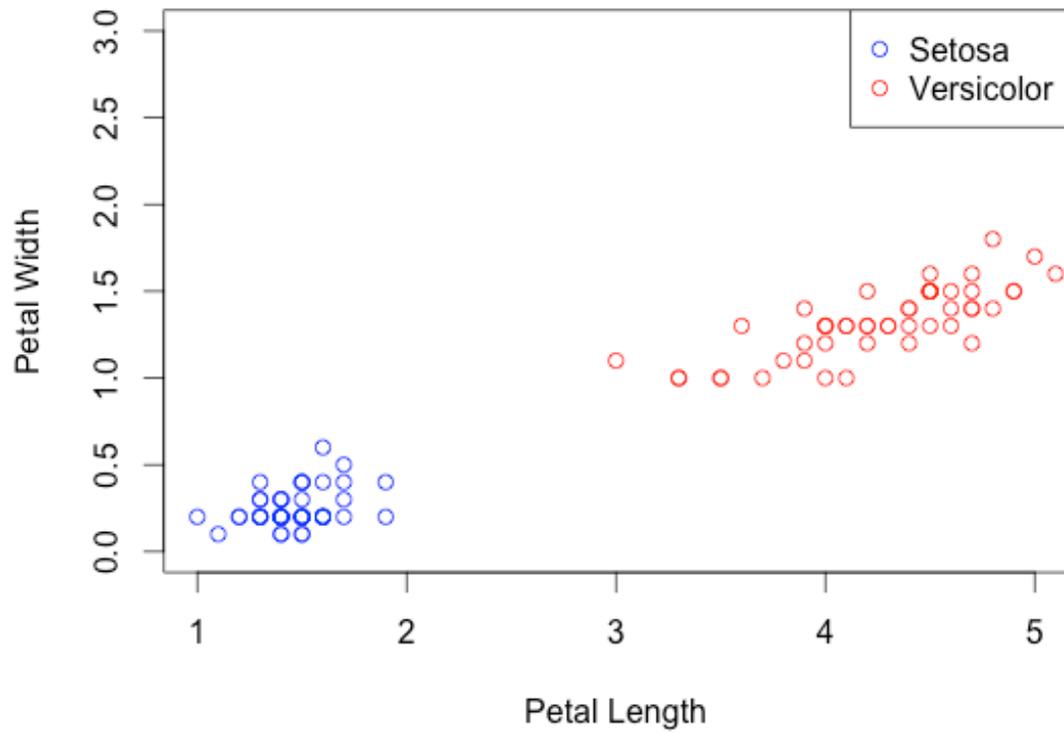
```
>par(mfrow=c(2,1))  
>plot(iris$Petal.Length)
```

```
>plot(iris$Petal.Width)
```



Multiple Sets of Data on the Same Plot & Legends

```
>par(mfrow=c(1,1))  
>plot(iris$Petal.Length[iris$Species=='setosa'],iris$Petal.Width[iris$Species=='setosa']  
, col='blue', xlim = c(1,5), ylim = c(0,3), xlab = "Petal Length", ylab = "Petal Width")  
>points(iris$Petal.Length[iris$Species=='versicolor'],iris$Petal.Width[iris$Species=='ve  
rsicolor'], col='red')  
> legend("topright", col = c('blue','red'), pch = 1, legend = c("Setosa", "Versicolor"))
```



There are additional packages that you can download and install such as 'ggplot2' to make more advanced graphs, if necessary!

Writing Functions

In addition, we can write our own function to calculate all of the descriptive statistics at once.

```
> summary.stat = function(x) {  
+   n = length(x)  
+   my.mean = mean(x)  
+   my.var = var(x)  
+   my.sd = sd(x)  
+   my.se = my.sd/sqrt(n)  
+   my.min = min(x)  
+   my.max = max(x)  
+   results = list(mean = my.mean, sd = my.sd, se = my.se, max = my.max, min = my.min,  
+ n = n)  
+   return(results)  
+ }
```

```
> summary.stat(beaver$Population)
```

```
$mean
```

```
[1] 410710.2
```

```
$sd
```

```
[1] 234351.5
```

```
$se
```

```
[1] 32815.78
```

```
$max
```

```
[1] 1071145
```

```
$min
```

```
[1] 59199
```

```
$n
```

```
[1] 51
```

Data Analysis – Simple Tests

2 Sample t-test

Goal: Compare the means of two independent samples, X1 and X2

$$H_0: \mu_1 = \mu_2$$

$$H_1: \mu_1 \neq \mu_2$$

$$H_2: \mu_1 < \mu_2$$

$$H_3: \mu_1 > \mu_2$$

Let's look at two species of iris from the iris dataset, and compare the means of the petal lengths.

First, subset the data for each of the two species.

```
> setosa <- subset(iris, iris$Species == "setosa")  
> versicolor <- subset(iris, iris$Species == "versicolor")
```

Next, use the t-test to compare the means.

First, use a 2-sided t-test to look at whether to accept the null hypothesis (means are equal) or the reject the null for the first alternate hypothesis (means are not equal).

2-sided t-test (alternative hypothesis 1)

```
> t.test(setosa$Petal.Length, versicolor$Petal.Length)
```

Welch Two Sample t-test

data: setosa\$Petal.Length and versicolor\$Petal.Length

t = -39.4927, df = 62.14, p-value < 2.2e-16

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

-2.939618 -2.656382

sample estimates:

mean of x mean of y

1.462 4.260

With a p-value of <2.2e-16 we can reject the null hypothesis, and conclude that mean petal length significantly differs between the two species (alternate hypothesis 1).

From here, we can then run one-sided t-tests to see if alternate hypotheses 2 or 3 is accepted.

1-sided t-test (alternative hypothesis 2 – less than)

```
> t.test(setosa$Petal.Length, versicolor$Petal.Length, alternative = "less")
```

Welch Two Sample t-test

```
data: setosa$Petal.Length and versicolor$Petal.Length
t = -39.4927, df = 62.14, p-value < 2.2e-16
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
 -Inf -2.679701
sample estimates:
mean of x mean of y
 1.462  4.260
```

1 sided t-test (alternative hypothesis 3 – greater than)

```
> t.test(setosa$Petal.Length, versicolor$Petal.Length, alternative = "greater")
```

Welch Two Sample t-test

```
data: setosa$Petal.Length and versicolor$Petal.Length
t = -39.4927, df = 62.14, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.916299  Inf
sample estimates:
mean of x mean of y
 1.462  4.260
```

The results show that the means petal lengths of the two species of iris are significantly different ($P < 2.2e-16$). The petal length of *Setosa* is significantly shorter than that of *Versicolor*. We accept hypothesis 2.

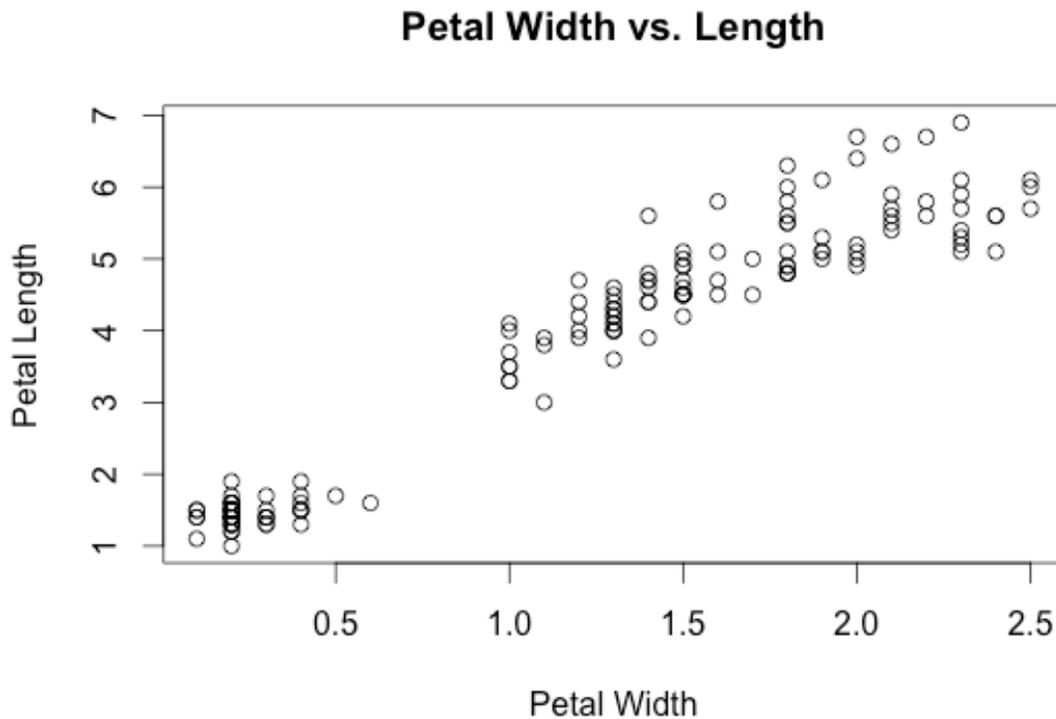
Note: The above analyses assume that the variances are unequal, to specify equal variance add `var.equal = TRUE` to the end of the argument.

Correlation

We will use the iris data to look at the relationship between petal length and width.

First, we can look at the relationship on a graph.

```
>plot(iris$Petal.Width, iris$Petal.Length, main = "Petal Width vs. Length", xlab =  
"Petal Width", ylab = "Petal Length")
```



The plot shows that there is a positive, linear relationship between petal length and width.

If we want to know if the relationship is significant, we can calculate the correlation between the two variables.

```
> cor.test(iris$Petal.Width, iris$Petal.Length)
```

Pearson's product-moment correlation

data: iris\$Petal.Width and iris\$Petal.Length

t = 43.3872, df = 148, p-value < 2.2e-16

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

0.9490525 0.9729853

sample estimates:

```
cor
0.9628654
```

Since the p-value is less than 0.05 there is a significant correlation between these two variables. The variables are 96% correlated.

Linear Regression (LR)

Goal of LR: Model linear relationship between an independent (X) and a dependent (Y) variable.

If we know the petal width can we estimate the petal length?

The simple linear regression model is:

$$y_i = \beta_0 + \beta_1 X + \epsilon_i$$

$\epsilon \sim normal(0, \sigma^2), iid$

First, fit a linear model.

```
> model <- lm(iris$Petal.Length ~ iris$Petal.Width)
```

Summarize the model.

```
> summary(model)
```

Call:

```
lm(formula = iris$Petal.Length ~ iris$Petal.Width)
```

Residuals:

```
   Min      1Q  Median      3Q      Max
-1.33542 -0.30347 -0.02955  0.25776  1.39453
```

Coefficients:

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.08356    0.07297   14.85 <2e-16 ***
iris$Petal.Width 2.22994    0.05140   43.39 <2e-16 ***
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.4782 on 148 degrees of freedom

Multiple R-squared: 0.9271, Adjusted R-squared: 0.9266

F-statistic: 1882 on 1 and 148 DF, p-value: < 2.2e-16

The results show us:

The intercept (β_0) = 1.08356

The slope (β_1) = 2.22994
The F statistic = 1882
P-value = <2.2e-16
R-squared = 0.9271

So, our predictive equation is:
Petal Length = 1.08356 + 2.22994(Petal Width)

These results tell us that:

- the intercept β_0 is significantly different than 0 ($P < 0.05$);
- the slope β_1 is significantly different than 0 ($P < 0.05$)
- the model is a good fit for the data ($R^2 = 0.9271$). This means that 93% of the variation in petal length is explained by the petal width.

We can also add a linear regression line directly in the plot of the two variables:

```
> plot(iris$Petal.Width, iris$Petal.Length, main = "Petal Width vs. Length", xlab =  
"Petal Width", ylab = "Petal Length")  
> abline(model)
```

