Lecture 3: Data Visualization

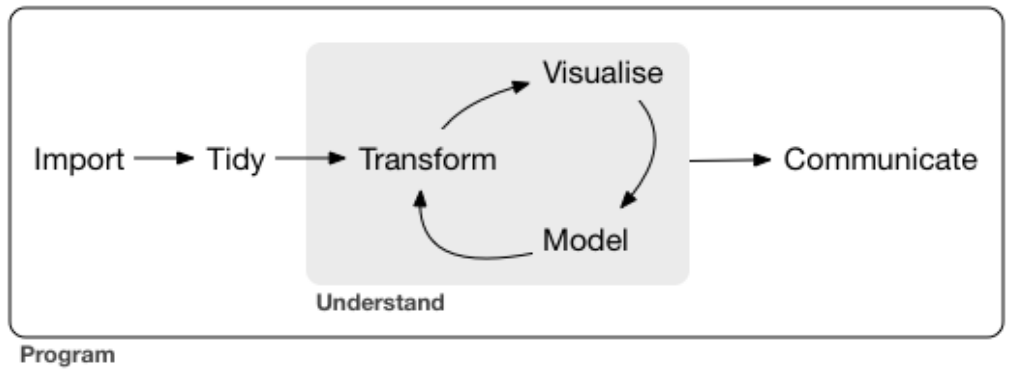
Importing data and Basic Graphics

Yujin Jeong

STATS 195

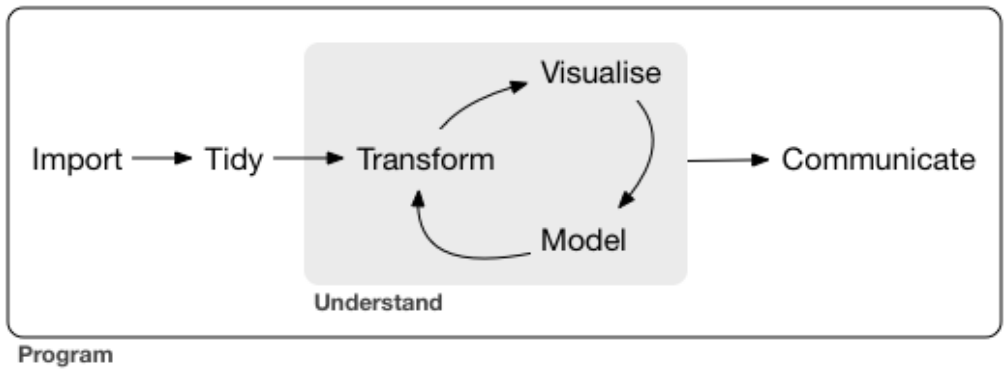
What you will learn

Our model of the tools needed in a typical data science project looks something like this:



What you will learn

Our model of the tools needed in a typical data science project looks something like this:

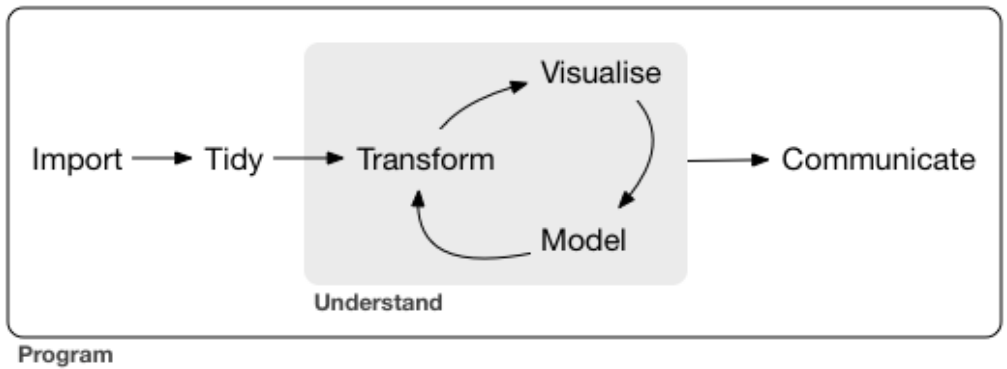


First you must **import** your data into R.

- This means that you take data stored in a file, database, or web application programming interface (API), and load it into a data frame in R.

What you will learn

Our model of the tools needed in a typical data science project looks something like this:

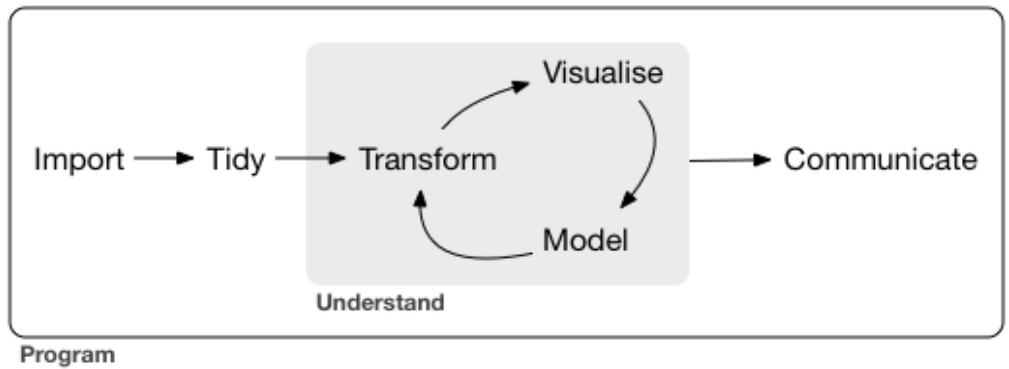


Once you've imported your data, it is a good idea to **tidy** it.

- Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored. In brief, when your data is tidy, each column is a variable, and each row is an observation.

What you will learn

Our model of the tools needed in a typical data science project looks something like this:

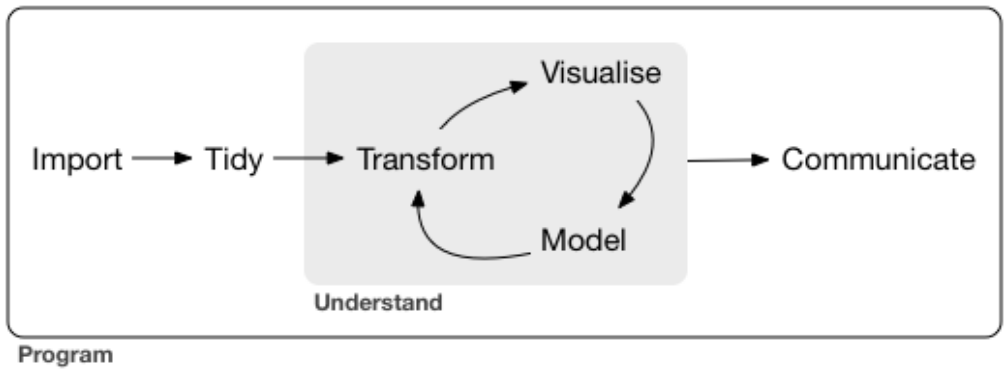


Once you have tidy data, a common first step is to **transform** it.

- Transformation includes narrowing in on observations of interest (like all people in one city, or all data from the last year), creating new variables that are functions of existing variables (like computing speed from distance and time), and calculating a set of summary statistics (like counts or means).

What you will learn

Our model of the tools needed in a typical data science project looks something like this:



Once you have tidy data with the variables you need, there are two main engines of knowledge generation: **visualisation** and **modelling**.

- A good visualisation will show you things that you did not expect, or raise new questions about the data. It might also hint that you're asking the wrong question, or you need to collect different data.
- Once you have made your questions sufficiently precise, you can use a model to answer them.

Importing Data

Recap: Data Frames

Dataframes are a special type of list in R. The values of the dataframe must all be vectors of the same length. You can think of each vector as a feature and each row as instance or subject in your dataset.

```
x <- runif(100)
y <- 5*x+2+rnorm(100)
z <- rep(c("Red","Blue","Yellow","Green"),25)
df1 <- data.frame(x,y,z)
str(df1)
```

```
## 'data.frame':  100 obs. of  3 variables:
##  $ x: num  0.266 0.372 0.573 0.908 0.202 ...
##  $ y: num  3.73 3.25 5.21 5.41 4.44 ...
##  $ z: chr  "Red" "Blue" "Yellow" "Green" ...
```

```
summary(df1)
```

```
##     height           width          color
## Min.   :0.01339   Min.   :0.8401   Length:100
## 1st Qu.:0.32308   1st Qu.:3.3835   Class :character
## Median :0.48781   Median :4.4865   Mode  :character
## Mean   :0.51785   Mean   :4.5717
## 3rd Qu.:0.76719   3rd Qu.:5.7630
## Max.   :0.99191   Max.   :8.4723
```

Tibbles

Tibbles are a special kind of dataframe. For our purposes, they are very similar but slightly easier to use, as they are more informative and can sometimes prevent mistakes.

```
install.packages("tibble")
library(tibble)
tb1 <- tibble(height = x,width = y,color = z)
str(tb1)
```

```
## tibble [100 × 3] (S3: tbl_df/tbl/data.frame)
##  $ height: num [1:100] 0.266 0.372 0.573 0.908 0.202 ...
##  $ width : num [1:100] 3.73 3.25 5.21 5.41 4.44 ...
 ## $ color : chr [1:100] "Red" "Blue" "Yellow" "Green" ...
```

```
summary(tb1)
```

```
##     height           width          color
## Min.   :0.01339   Min.   :0.8401   Length:100
## 1st Qu.:0.32308   1st Qu.:3.3835   Class :character
## Median :0.48781   Median :4.4865   Mode  :character
## Mean   :0.51785   Mean   :4.5717
## 3rd Qu.:0.76719   3rd Qu.:5.7630
## Max.   :0.99191   Max.   :8.4723
```

Factors

Factor data is similar to character data, but the allowable values are fixed. Let's use our example from above.

```
z2 <- factor(z)
str(z2)
```

```
## Factor w/ 4 levels "Blue","Green",..: 3 1 4 2 3 1 4 2 3 1 ...
```

In this case, factor figured out what levels to use by looking at all the unique values in z. We could have done this explicitly:

```
z3 <- factor(z,levels=c("Red", "Blue", "Yellow", "Green"))
str(z3)
```

```
## Factor w/ 4 levels "Red","Blue","Yellow",..: 1 2 3 4 1 2 3 4 1 2 ...
```

Factors

Factors will force us to be more careful when we add new rows or make changes.

```
z[2] <- "Bule"
str(z)
```

```
##  chr [1:100] "Red" "Bule" "Yellow" "Green" "Red" "Blue" "Yellow" "Green" "Red" "Blue" "Yellow" ...
```

```
z3[2] <- "Bule"
str(z3)
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "Bule") :
##   invalid factor level, NA generated
##  Factor w/ 4 levels "Red","Blue","Yellow",..: 1 NA 3 4 1 2 3 4 1 2 ...
```

Factors

We can fix our tibble above to use a factor for color, you'll see it also makes our summary output more useful.

```
tb1$color <- factor(tb1$color)
summary(tb1)
```

```
##     height            width           color
## Min.   :0.01339   Min.   :0.8401   Blue  :25
## 1st Qu.:0.32308   1st Qu.:3.3835   Green :25
## Median :0.48781   Median :4.4865   Red   :25
## Mean   :0.51785   Mean   :4.5717   Yellow:25
## 3rd Qu.:0.76719   3rd Qu.:5.7630
## Max.   :0.99191   Max.   :8.4723
```

Importing Data

If we want to read a file from the internet or locally as a dataframe, we can use the read functions in R. Typically files will be comma separated (csv) or tab separated. We can use the general function `read.delim()` or specific functions like `read.csv()`.

```
iris_df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=F)
summary(iris_df)
```

```
##       V1              V2              V3              V4            V5
## Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   Length:150
## 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   Class :character
## Median :5.800   Median :3.000   Median :4.350   Median :1.300   Mode  :character
## Mean   :5.843   Mean   :3.054   Mean   :3.759   Mean   :1.199
## 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
## Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

We see some issues with the data. Let's check the source website to see if there is more information.

Importing Data

Based on the website, we can clean things up a little bit:

```
names(iris_df) <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")
iris_df$Species <- factor(iris_df$Species)
summary(iris_df)
```

```
##  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
## Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   Iris-setosa    :50
## 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   Iris-versicolor:50
## Median :5.800   Median :3.000   Median :4.350   Median :1.300   Iris-virginica :50
## Mean   :5.843   Mean   :3.054   Mean   :3.759   Mean   :1.199
## 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
## Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Importing Data

The tidyverse package for reading files is called readr. It works much the same what as base R and makes tibbles by default.

Its functions are written with underscores replacing the periods e.g. `read_csv()` instead of `read.csv()`.

```
#install.packages("readr")
library(readr)
iris_tb <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", col_names = FALSE)
```

The results are again very similar although you will notice that the read function has a bit of useful output that it provides as you import the data.

We can clean the data as before.

```
names(iris_tb) <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")
iris_tb$Species <- factor(iris_tb$Species)
```

Basic Graphics

We will cover three base R graphical functions: "plot", "hist", and "boxplot" in this lecture and give much more focus to tidyverse’s graphics package "ggplot2" in the next lecture.

Histograms

The function `hist` computes a histogram of the given data values.

```
hist(iris_tb$Sepal.Length)
```

Histograms

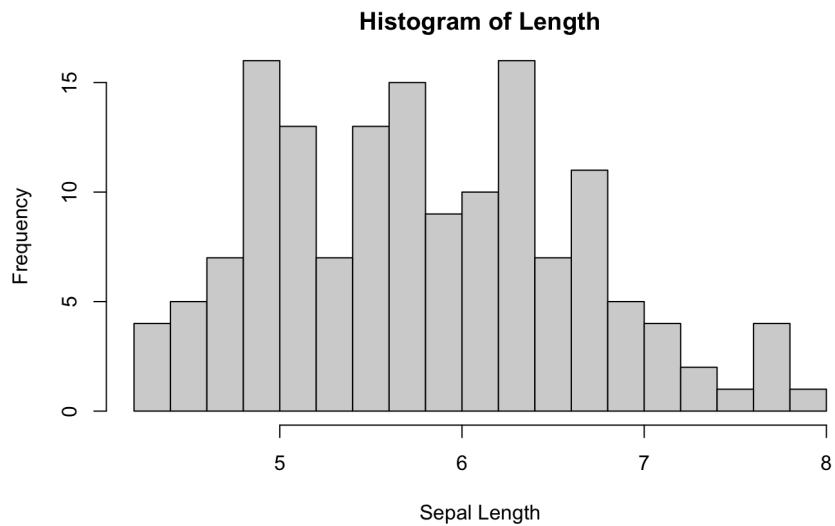
Let’s add some optional arguments to make the histogram easier to read.

```
hist(iris_tb$Sepal.Length, main="Histogram of Length", xlab="Sepal Length")
```

Histograms

We can change the number of bins/breaks either by giving it a number of breaks or by telling it exactly where the breaks should be. Note: You can also use a specific algorithm to decide how many and where they should be (by default it's the Sturges algorithm), but this discussion is beyond the scope of this class.

```
hist(iris_tb$Sepal.Length, breaks=20, main="Histogram of Length", xlab="Sepal Length")
```



Histograms

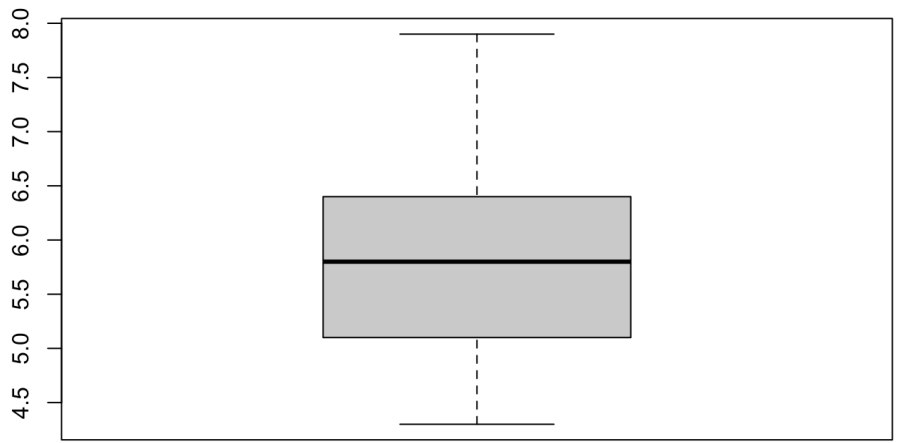
We can change the y-axis to be a density instead of frequency.

```
hist(iris_tb$Sepal.Length, breaks=8, probability=TRUE, main="Histogram of Length", xlab="Sepal Length")
```

Boxplots

The function `boxplot` produces a boxplot of the given data values.

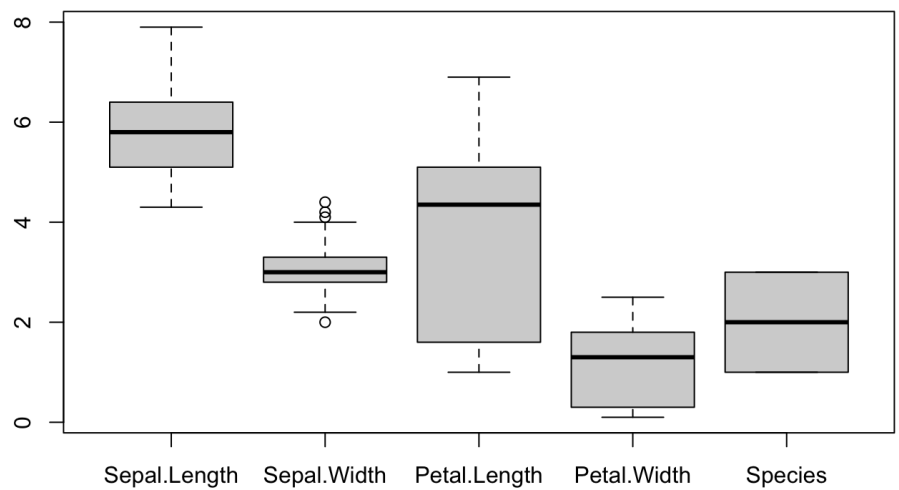
```
boxplot(iris_tb$Sepal.Length)
```



Boxplots

It can be useful for comparing across related variables.

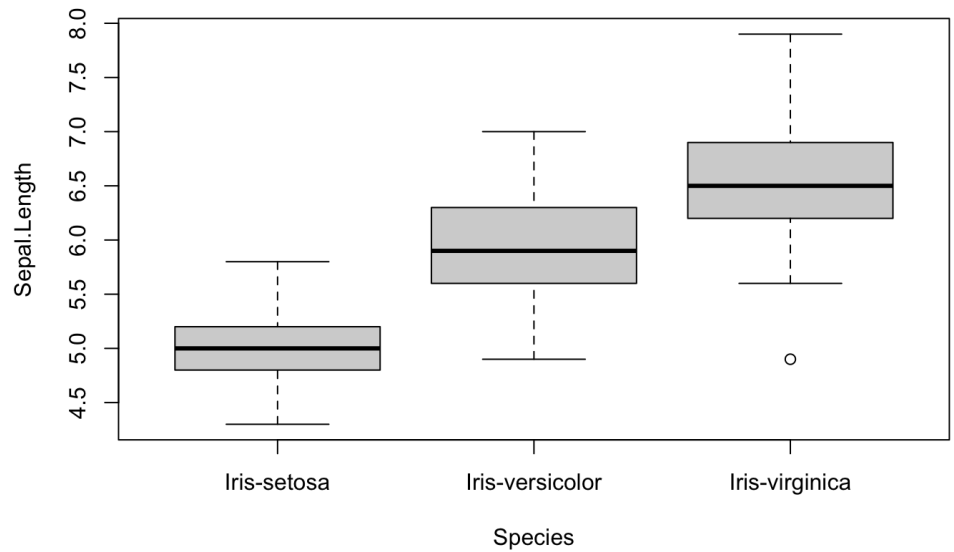
```
boxplot(iris_tb)
```



Boxplots

It can also be useful for viewing the relationship between a quantitative variable and a qualitative variable.

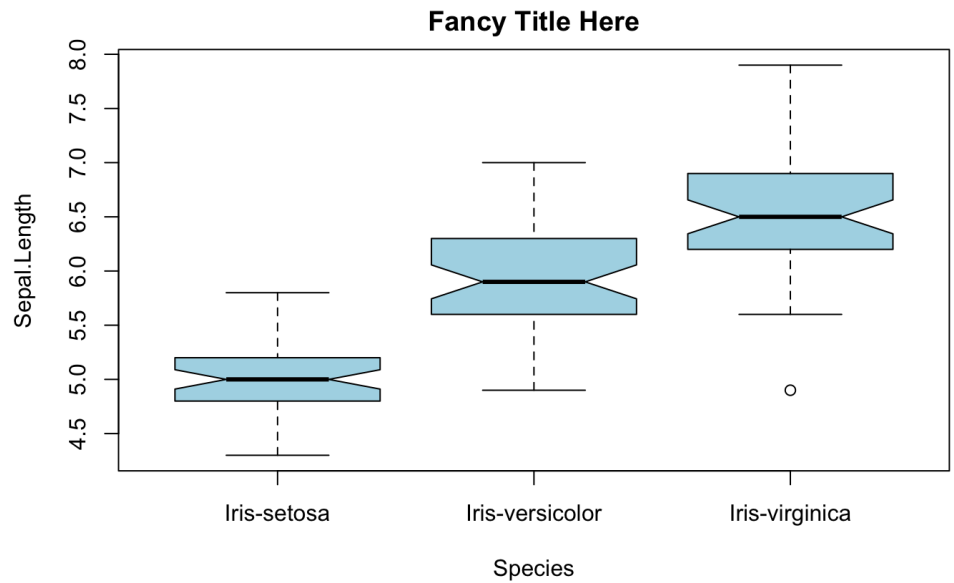
```
boxplot(Sepal.Length~Species, data=iris_tb)
```



Boxplots

Finally, we can add nice pretty details including notches (which historically have been used to visually compare the medians of different bloxplots).

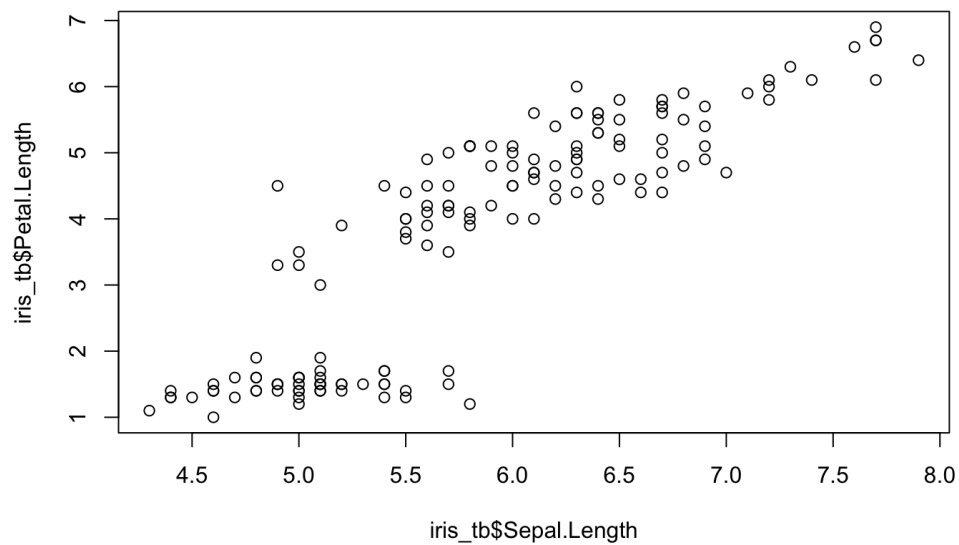
```
boxplot(Sepal.Length~Species, data=iris_tb, main="Fancy Title Here", col="lightblue", notch=TRUE)
```



Scatter Plots

If we want to look at the relationship between two quantitative variables, we can use a scatter plot.

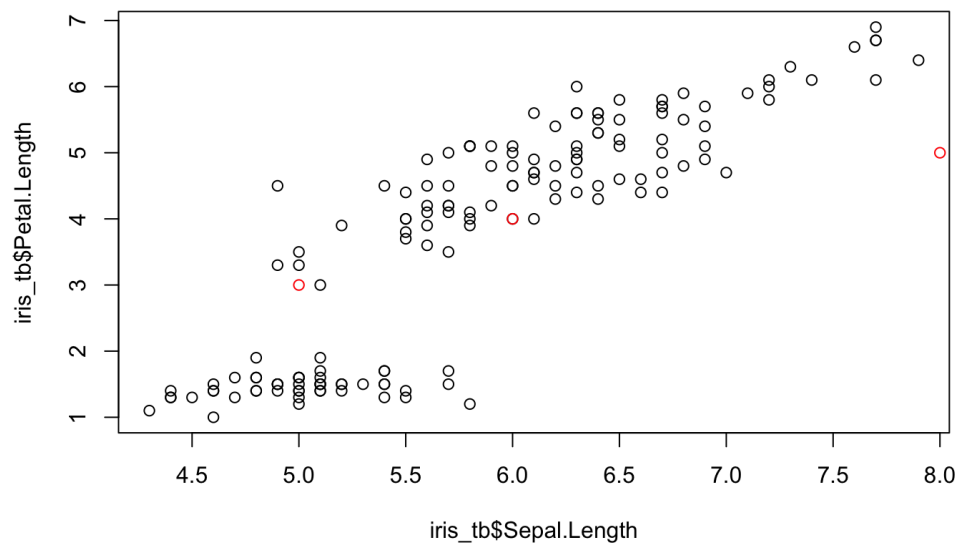
```
plot(iris_tb$Sepal.Length,iris_tb$Petal.Length)
```



Scatter Plots

The function `points` would let you overlay multiple scatter plots with the axes from the original plot.

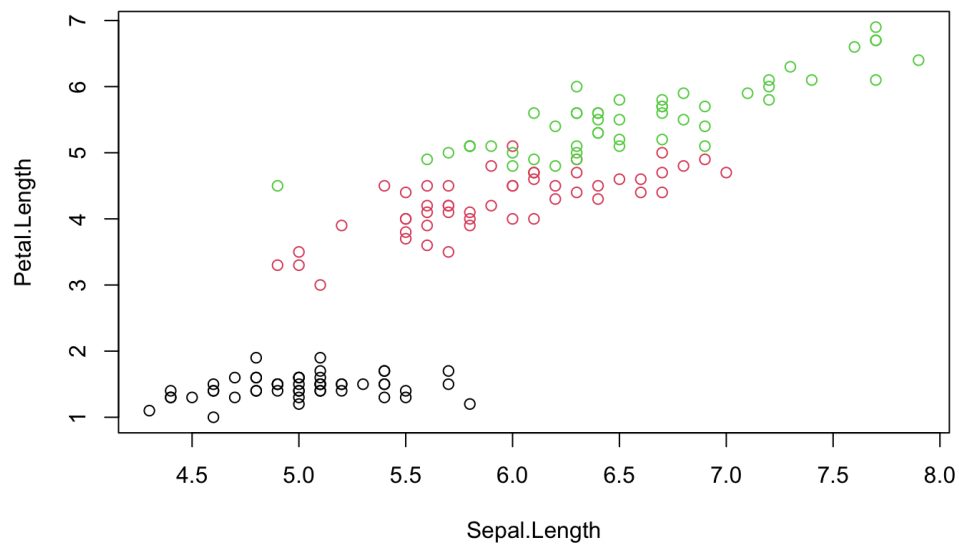
```
plot(iris_tb$Sepal.Length, iris_tb$Petal.Length)
points(x=c(5,6,8,9), y=c(3,4,5,6), col="red")
```



Scatter Plots

We can color our points using factors.

```
plot(Petal.Length~Sepal.Length, data=iris_tb, col=Species)
```



Scatter Plots

We can also draw lines on top of our plots using the `abline` function. Note that "lty" controls line types.

```
plot(x, y, main="Toy Example")
abline(a=2, b=5, col="red")
abline(h=8, col="purple", lty=2)
abline(v=0.5, lty=5)
```