(A Brief) Introduction to R

Matt Carson, matthew.carson@northwestern.edu

May 15, 2018

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. Writing code with the Markdown framework helps to facilitate organization and reproducibility in your research and makes it easy to share nicely-formatted reports with your colleges. For more details on using R Markdown see *http://rmarkdown.rstudio.com*.

A **reference guide** is here: *https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf*

A **cheat sheet** is avaiable at *https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf*.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

See *https://www.r-bloggers.com/r-markdown-and-knitr-tutorial-part-1/* and *https://www.r-bloggers.com/r-markdown-and-knitr-tutorial-part-2/* for **detailed instructions on how to set up Knitr in R Studio**.

If you are interested in becoming more proficient in the **R Studio environment**, take this 3-hour Data Camp course: *https://www.datacamp.com/courses/working-with-the-rstudio-ide-part-1*

NOTE:

Several of the code chunks that we will execute have R Studio interface equivalents. I will point these out in the text below where applicable in a block like this:

> — There's an R Studio feature for this!!! —

The Very Basics...

R is a high-level, interpreted programming language and environment for (primarily) statistical analysis. Read more about it here: *https://www.r-project.org/about.html*.

We will start by going through the fundamental stuff.

Arithmetic Operations

```
# Addition
5 + 6
```

```
## [1] 11
```

```
# Subtraction
100 - 1
```

```
## [1] 99
```

```r
# Multiplication
4 * 4
```

```
## [1] 16
```

```r
# Division
(6 - 5) / 4
```

```
## [1] 0.25
```

```r
# Exponentiation
2 ^ 100
```

```
## [1] 1.267651e+30
```

```r
# Modulo (returns the remainder)
21 %% 5
```

```
## [1] 1
```

Assigning variables

```r
# The '<-' operator assigns a value to a variable
x <- 5 + 6
x
```

```
## [1] 11
```

```r
# However, '=' also works.
y = 100 - 1
print (paste("y equals", y))
```

```
## [1] "y equals 99"
```

```r
z <- x + y
print (paste("z equals x + y, or", z))
```

```
## [1] "z equals x + y, or 110"
```

Basic Data Types

The following are basic data types in R:

- numeric (real numbers like 5.3)
- integer (1, 2, 3…also numeric class)
- character (text)
- logical (Boolean values, TRUE and FALSE)

```r
# Numeric
numeric_value <- 234.77

# The function class() returns the data type for the R object
print(paste("numeric_value", numeric_value, "is of type", class (numeric_value)))
```

```
## [1] "numeric_value 234.77 is of type numeric"
```

```r
# Integer
integer_value <- 234
```

```r
print(paste("integer_value", integer_value, "is of type", class (integer_value)))
```

```
## [1] "integer_value 234 is of type numeric"
```

```r
# Character
# By putting quotes around the value, we are telling R that this is text.
character_value <- "234"

print(paste("character_value", character_value, "is of type", class (character_value)))
```

```
## [1] "character_value 234 is of type character"
```

```r
# Logical
logical_value <- TRUE

print(paste("logical_value", logical_value, "is of type", class (logical_value)))
```

```
## [1] "logical_value TRUE is of type logical"
```

Vectors

In R, a **vector** is a one-dimensional array. It is a sequence of elements of the same data type. Elements of a vector are called **components**. You assign elements to a vector using the **combine function**, *c()*.

NOTE: Technically, everything in R is a vector. You don't need to worry about this as a beginner user, so we won't go into it here. If you want to go into the weeds on this, here are a couple of blogs discussing this subject:

- *http://www.noamross.net/blog/2014/4/16/vectorization-in-r--why.html*
- *http://alyssafrazee.com/2014/01/29/vectorization.html*

Vector Types

```r
# Different types of vectors
numeric_vector <- c(12, 76, 245)
numeric_vector
```

```
## [1]  12  76 245
```

```r
character_vector <- c("twelve", "seventy six", "two hundred forty five")
character_vector
```

```
## [1] "twelve"                 "seventy six"
## [3] "two hundred forty five"
```

```r
logical_vector <- c(TRUE, TRUE, FALSE)
logical_vector
```

```
## [1]  TRUE  TRUE FALSE
```

Naming Vectors

```r
# Before names are added...
num_guitars <- c(1, 4, 2, 15)
num_guitars
```

```
## [1]  1  4  2 15
```

```r
# After...
names(num_guitars) <- c("Bob", "Sue", "Max", "Emilia")
num_guitars
```

```
##    Bob    Sue    Max Emilia
##      1      4      2     15
```

```r
# How many guitars does Emilia have?
num_guitars["Emilia"]
```

```
## Emilia
##     15
```

```r
num_guitars[4]
```

```
## Emilia
##     15
```

```r
# You could also assign the names like so...
num_guitars <- c(1, 4, 2, 15)
guitar_names <- c("Bob", "Sue", "Max", "Emilia")

names(num_guitars) <- guitar_names
num_guitars
```

```
##    Bob    Sue    Max Emilia
##      1      4      2     15
```

Adding Vectors

```r
# Vector describing the number of guitars owned by four people
num_guitars <- c(1, 4, 2, 15)
names(num_guitars) <- c("Bob", "Sue", "Max", "Emilia")

# How many guitars do they own collectively?
total_guitars <- sum(num_guitars)
print(paste("Total number of guitars:", total_guitars))
```

```
## [1] "Total number of guitars: 22"
```

```r
# Bob, Sue, Max, and Emilia went to a guitar auction. Some bought and some sold guitars.
bought_and_sold <- c(-1,-2, 8, 0)

# How many does each person own now?
new_num_guitars <- num_guitars + bought_and_sold
new_num_guitars
```

```
##    Bob    Sue    Max Emilia
##      0      2     10     15
```

Notice that both vectors don't have to be named; R will perform the operation element-wise and transfer the names to the new vector, 'new_num_guitars'. But what if both vectors have names and they aren't the same?

```
# Vector describing the number of guitars owned by four people
num_guitars <- c(1, 4, 2, 15)
names(num_guitars) <- c("Bob", "Sue", "Max", "Emilia")

# Bob, Sue, Max, and Emilia went to a guitar auction. Some bought and some sold guitars.
bought_and_sold <- c(-1,-2, 8, 0)
names(bought_and_sold) <- c("Joe", "Frank", "Lisa", "John")

# Which set of names get transferred to the new vector?
new_num_guitars <- num_guitars + bought_and_sold
new_num_guitars
```

```
##    Bob    Sue    Max Emilia
##      0      2     10     15
```

```
# How about if we switch the order on the right side?
new_num_guitars <- bought_and_sold + num_guitars
new_num_guitars
```

```
##   Joe Frank  Lisa  John
##     0     2    10    15
```

The set of names associated with the left-most vector in the addition operation is the one chosen. Be aware of this if you are performing operations on named R objects!

Matrices

A matrix is a 2-dimensional vector (i.e., it has rows and columns) that contains elements of the same type. In R, these can be numeric, character, or logical values.

```
# A matrix with 4 rows with the number 1 - 12 by row
matrix(1:12, byrow=TRUE, nrow=4)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

```
# You can also create a matrix like this:
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- c(7, 8, 9)
d <- c(10, 11, 12)

mat <- matrix(c(a, b, c, d), nrow = 4)

# Vectors to name matrix rows and columns
colors <- c("red", "green", "blue")
shapes <- c("circle", "square", "rectangle", "triangle")

# These functions add names to rows and columns of a matrix
```

```r
colnames(mat) <- colors
rownames(mat) <- shapes

# 'shapes' matrix with row and column names:
mat
```

```
##           red green blue
## circle      1     5    9
## square      2     6   10
## rectangle   3     7   11
## triangle    4     8   12
```

```r
# Find the total numbers of each shape in the matrix
total <- rowSums(mat)

# Bind the new vector as a column to mat
tot_mat <- cbind(mat, total)

# Same matrix but with an added column of row totals
tot_mat
```

```
##           red green blue total
## circle      1     5    9    15
## square      2     6   10    18
## rectangle   3     7   11    21
## triangle    4     8   12    24
```

```r
# Add a row
hexagon <- c(13, 14, 15, 42)

tot_mat <- rbind(tot_mat, hexagon)

# Now with an added row
tot_mat
```

```
##           red green blue total
## circle      1     5    9    15
## square      2     6   10    18
## rectangle   3     7   11    21
## triangle    4     8   12    24
## hexagon    13    14   15    42
```

```r
# Select just the circle row
circles <- mat[1,]
circles
```

```
##   red green  blue
##     1     5     9
```

```r
# How many green squares and rectangles are there?
some <- mat[2:3,2]

# Green squares and rectangles
some
```

```
##    square rectangle
##         6         7
```

```r
# Average number of shapes
mean(mat)
```

```
## [1] 6.5
```

```r
# Subtract 2 from all elements
mat - 2
```

```
##           red green blue
## circle     -1     3    7
## square      0     4    8
## rectangle   1     5    9
## triangle    2     6   10
```

Factors

In R, categorical variables are called factors. These variables can be composed of either unordered categories or ordered levels (ordinal).

```r
# Gender, a non-ordered categorical variable of type 'character'
genders <- c("Male", "Female", "Female", "Male", "Female")
genders
```

```
## [1] "Male"   "Female" "Female" "Male"   "Female"
```

```r
# Change 'genders' to a factor
factor_genders <- factor(genders)
factor_genders
```

```
## [1] Male   Female Female Male   Female
## Levels: Female Male
```

```r
# Size, an ordinal variable
size <- c("Small", "Large", "Medium", "Medium", "Large", "Large", "Small")
factor_size <- factor(size, ordered = TRUE, levels = c("Small", "Medium", "Large"))
factor_size
```

```
## [1] Small  Large  Medium Medium Large  Large  Small
## Levels: Small < Medium < Large
```

```r
# Give a summary of the counts of each category
summary(factor_size)
```

```
##  Small Medium  Large
##      2      2      3
```

```r
# Choose a size to compare (the first component of the vector)
first_size <- factor_size[1]

# Choose another size (the fourth component of the vector)
second_size <- factor_size[4]

# Is the first size larger than the second size?
print(first_size)
```

```
## [1] Small
## Levels: Small < Medium < Large
```

```
print(second_size)
```

```
## [1] Medium
## Levels: Small < Medium < Large
```

```
print(paste("Is the first size larger than the second size?", first_size > second_size))
```

```
## [1] "Is the first size larger than the second size? FALSE"
```

We'll see more examples of factors later, so let's move on.

Data Frames

Like a matrix, a data frame has rows and columns. However, unlike a matrix, a data frame can contain multiple data types. Columns contain **variables** and rows contain **observations**. The structure of a data frame is similar to that of an Excel spreadsheet.

```
# Create vectors containing data about four people's guitar collections
owner <- c("Bob", "Sue", "Max", "Emilia")
count <- c(0, 2, 10, 15)
value <- c(0, 605.50, 5140.10, 9243.44)
professional <- c(FALSE, FALSE, TRUE, TRUE)

# Create a data frame from the vectors
guitars_df <- data.frame(owner, count, value, professional)
guitars_df
```

```
##    owner count   value professional
## 1    Bob     0    0.00        FALSE
## 2    Sue     2  605.50        FALSE
## 3    Max    10 5140.10         TRUE
## 4 Emilia    15 9243.44         TRUE
```

```
# Returns only the 'professional' variable
#guitars_df$professional

# Returns the entire row for every professional player
# Use ! to return all non-professiona players
#guitars_df[professional,]
```

Lists

Lists let you store many types of R objects in one structure. These objects don't have to be related at all, but often are. A common use of lists is to provide a single container for different data types returned by a single R function. We'll see an example of this a little later.

```
# Vector of 5 numbers
v <- 1:5

# A 3 x 2 matrix of numbers from 1 to 6
m <- matrix(1:6, ncol = 2)

# First 5 rows of the built-in data set 'iris'
iris_head <- iris[1:5,]
```

```
# Create a list and give the component names
some_list <- list(vec = v,
                  mat = m,
                  df  = iris_head)
some_list
```

```
## $vec
## [1] 1 2 3 4 5
## 
## $mat
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
## 
## $df
##   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
```

```
# Print the vector component of some_list
# All three of these do the same thing
some_list$vec # by name
```

```
## [1] 1 2 3 4 5
```

```
some_list[["vec"]] # using a text value for the name
```

```
## [1] 1 2 3 4 5
```

```
some_list[[1]] # by index
```

```
## [1] 1 2 3 4 5
```

```
# Print the second element of 'vec'
some_list[["vec"]][2]
```

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

```
# Add an element to a list
some_list$another_component <- c("a", "b", "c")
some_list
```

```
## $vec
## [1] 1 2 3 4 5
## 
## $mat
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
## 
## $df
##   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
##
## $another_component
## [1] "a" "b" "c"
```

These are the very basics of R. We will now go through an example where we explore a data set and calculate a few descriptive statistics.

Working with a data set

First, how to get help from R

We are going to work with one of R's built-in data sets called *iris*. First, let's find out more about the iris data. We can do that by typing '?' before the name, or by typing 'help (iris)'.

```
# What is 'iris'?
?iris

# All of these do the same thing; quotes are optional
# help(iris)
# help("iris")
# ?"iris"
```

More information about getting help in R (including vignettes, codes demonstrations, and searching) can be found here: *https://www.r-project.org/help.html*.

> — You can also search for terms using the **'Help'** tab on the right. —

R Bloggers (*https://www.r-bloggers.com/*) and Stack Exchange's **Cross Validated** (*https://stats.stackexchange.com/*) are also great resources for examples and explanations of both statistical concepts and R code.

Getting to know the data

What does the *iris* data set look like? You can use the *View()* function to get an 'Excel-like' view of your data:

```
View(iris)
```

> — You can also do this by double clicking on the object name in the **'Environment'** window in the upper right corner. You can sort each column by clicking on the column name. —

When working with a new data set, it's a good idea to take a few minutes to examine and summarize it. Let's take a look at *iris*. The class function tells us the type of object we are dealing with.

```
# Slowly type 'class(iris)' below
class(iris)
```

```
## [1] "data.frame"
```

As you typed the ‘class’ command, you should have seen a pop-up menu of available functions and their descriptions. As you continue to type *iris*, you will see a description of available objects and their descriptions that match the name *iris*.

You can check the structure of *iris* to see if the variable types seem correct. This is done using the *str()* function.

> — You can also click the blue circle next to the appropriate object under the **‘Environment’** tab on the right —

```
# str() returns the structure of the object
str(iris)
```

```
## '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 ...
```

Notice that the first four variables are of type ‘numeric’. The fifth variable, ‘Species’, is of type ‘factor’. In R, categorical variables are called **factors**. *R will set imported string as factors unless you tell it otherwise.* We’ll see how to do this later.

What is the size of the *iris* data set? The *dim()* command will tell us.

```
# dim() returns the dimensions of the data set
dim(iris)
```

```
## [1] 150   5
```

The output confirms that there are 150 rows and 5 columns.

You can check the column (variable) names for your data set with *names()*:

```
# names() returns the names of the variables (columns) in the data set
names(iris)
```

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

What if we want basic counts and distribution characterisitics for all variables? We can use *summary()* for this:

```
# summary() returns a summary of all data in the object
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   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
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
summary(iris$Sepal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

```
fivenum(iris$Sepal.Length)
```

```
## [1] 4.3 5.1 5.8 6.4 7.9
```

summary() returns the appropriate statistics for each variable type. For numeric values, it returns a 5-number summary along with the mean. For catergorical variables (factors), it returns a count for each category.

Another handy set of functions is *head()* and *tail()*. They return the first and last *n* rows of a data frame, respectively:

```
# The first n rows...
head(iris, n = 5)
```

```
##   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
```

```
# The last n rows...
tail(iris, n = 7)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 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
```

You can also print the entire contents of an object with *print()*, although you should be careful about printing large objects to the screen.

```
# WARNING!!!
#
# Not recommended for large data sets
#print(iris)
```

Subsetting the data

Let's analyze just the petal widths of the versicolor species in *iris*. We can create a subset of the data like so:

```
# We can create a new data frame by extracting a subet of all versicolor records,...
versicolor <- iris[iris$Species == 'versicolor',]
versicolor
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 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
```

```
# ...then create a numeric vector of versicolor petal widths, or...
versicolor_pw <- versicolor$Petal.Width
versicolor_pw
```

```
##  [1] 1.4 1.5 1.5 1.3 1.5 1.3 1.6 1.0 1.3 1.4 1.0 1.5 1.0 1.4 1.3 1.4 1.5
## [18] 1.0 1.5 1.1 1.8 1.3 1.5 1.2 1.3 1.4 1.4 1.7 1.5 1.0 1.1 1.0 1.2 1.6
## [35] 1.5 1.6 1.5 1.3 1.3 1.3 1.2 1.4 1.2 1.0 1.3 1.2 1.3 1.3 1.1 1.3
```

```
# ...do the same thing all in one line
versicolor_pw <- iris[iris$Species == 'versicolor', 'Petal.Width']
versicolor_pw
```

```
##  [1] 1.4 1.5 1.5 1.3 1.5 1.3 1.6 1.0 1.3 1.4 1.0 1.5 1.0 1.4 1.3 1.4 1.5
## [18] 1.0 1.5 1.1 1.8 1.3 1.5 1.2 1.3 1.4 1.4 1.7 1.5 1.0 1.1 1.0 1.2 1.6
## [35] 1.5 1.6 1.5 1.3 1.3 1.3 1.2 1.4 1.2 1.0 1.3 1.2 1.3 1.3 1.1 1.3
```

As with most programming languages, there is usually more than one way to do things in R. Keep this in mind as you are learning the language.

Calculating simple statistics

Let's get some simple stats on the pedal widths of versicolor:

```
mean(versicolor_pw)
```

```
## [1] 1.326
```

```
median(versicolor_pw)
```

```
## [1] 1.3
```

```
var(versicolor_pw)
```

```
## [1] 0.03910612
```

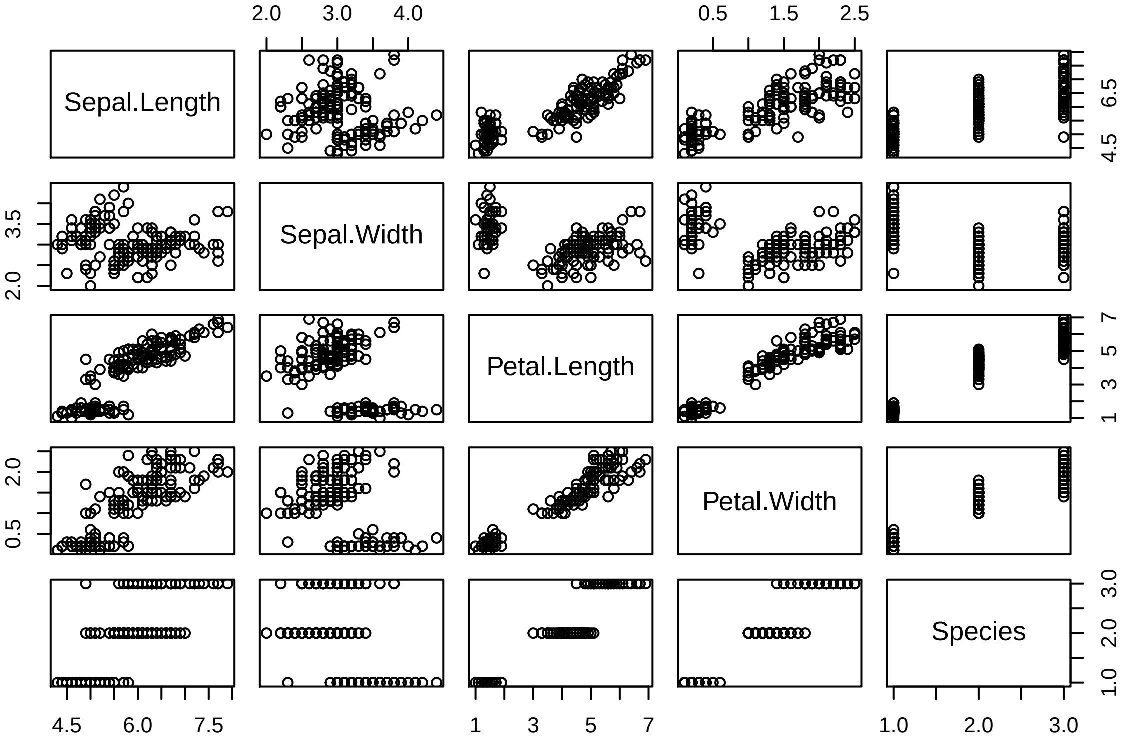
```
sd(versicolor_pw)
```

```
## [1] 0.1977527
```

Plotting Data

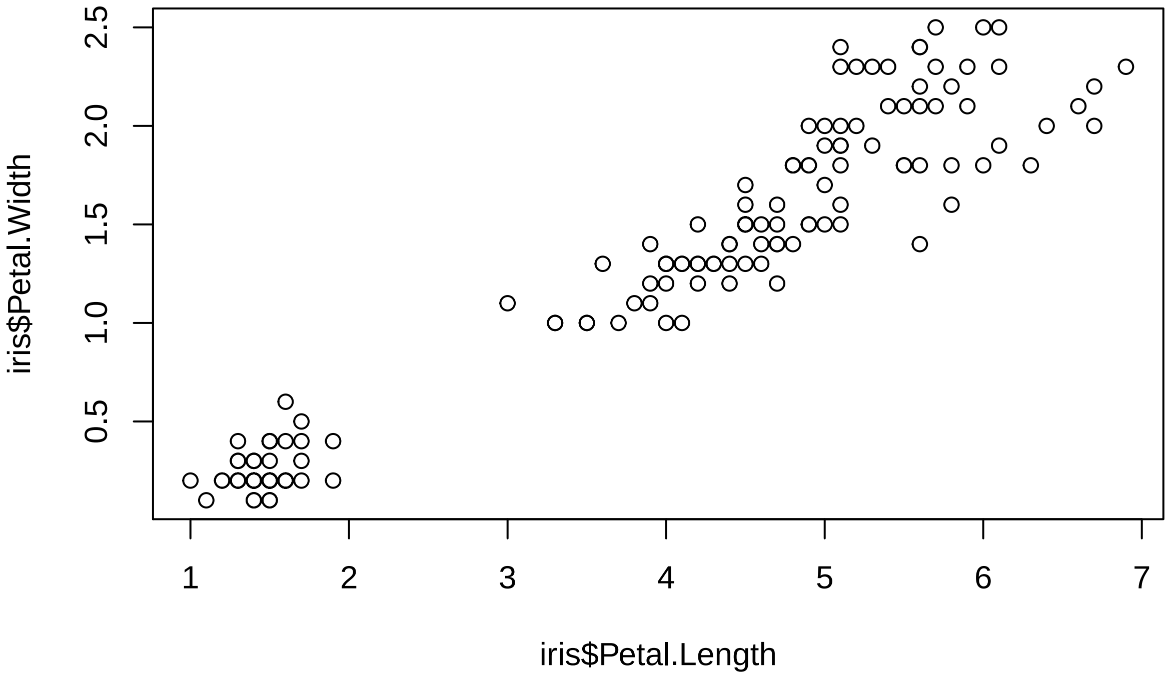
What happens if we plot the whole *iris* data set?

```
plot(iris)
```



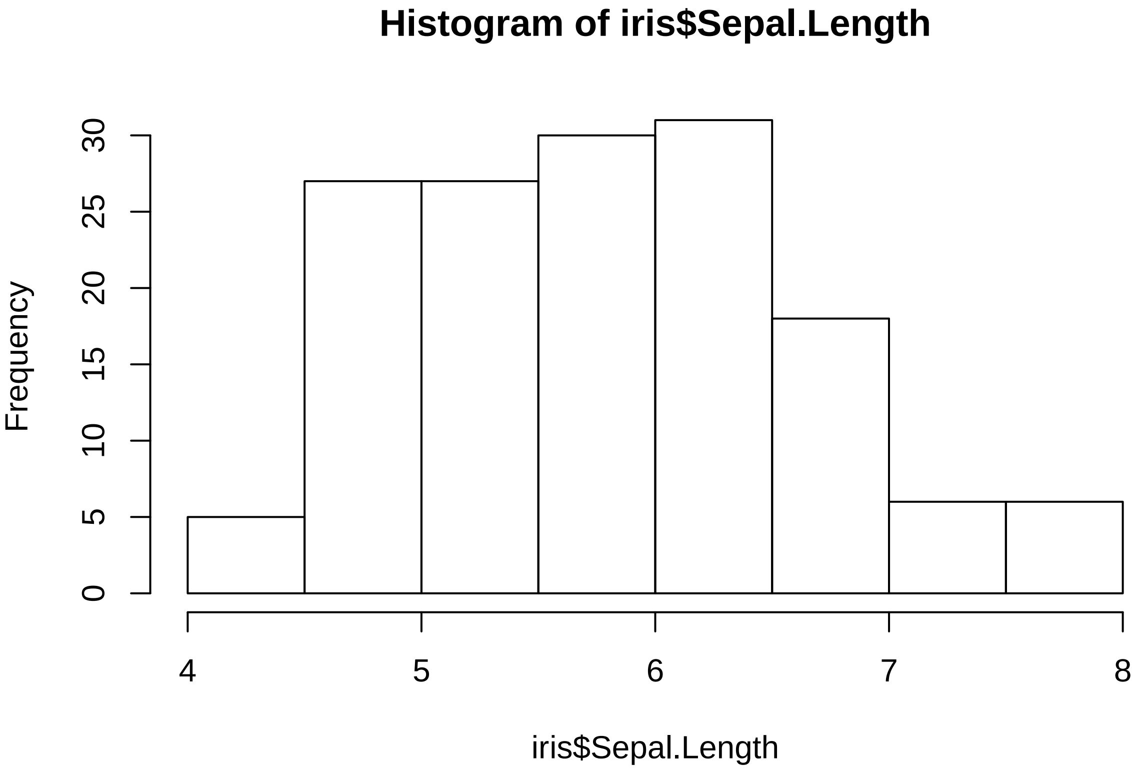
Calling *plot()* on the whole data set makes a scatterplot matrix that plots every variable against every other variable. What if we just want to look at one of these scatterplots?

```
plot(iris$Petal.Length, iris$Petal.Width)
```



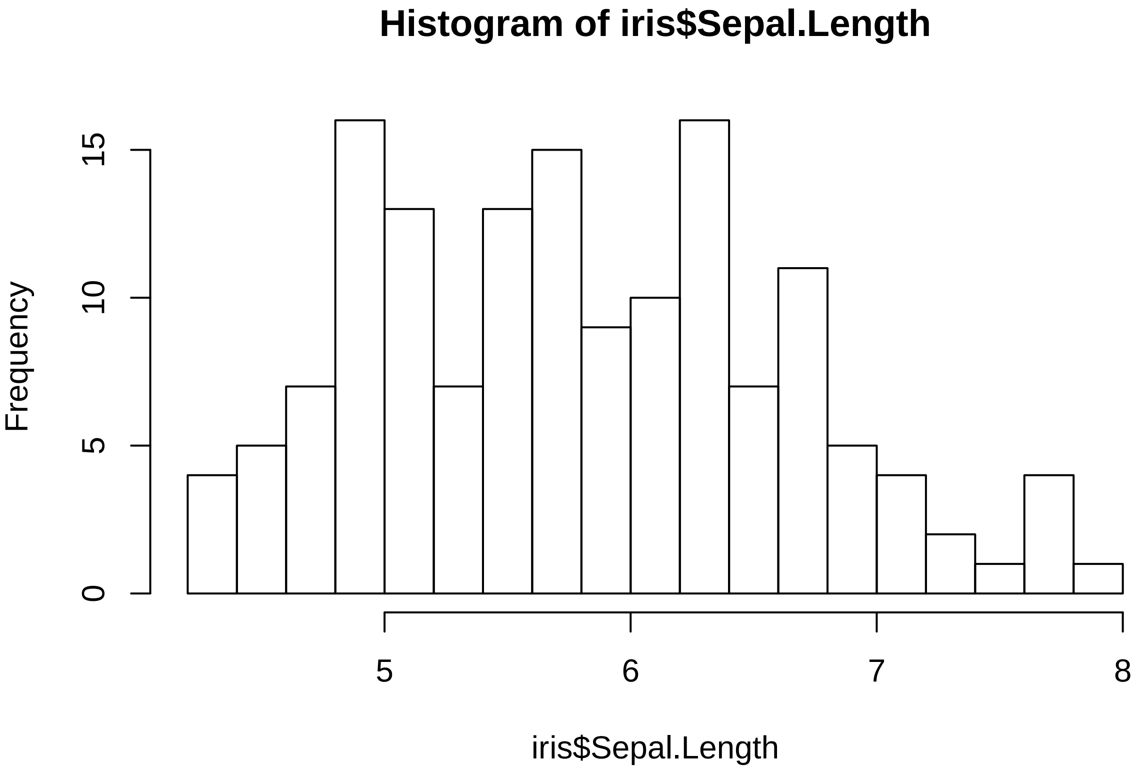
We can also make a histogram that describes the distribution of sepal length values:

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



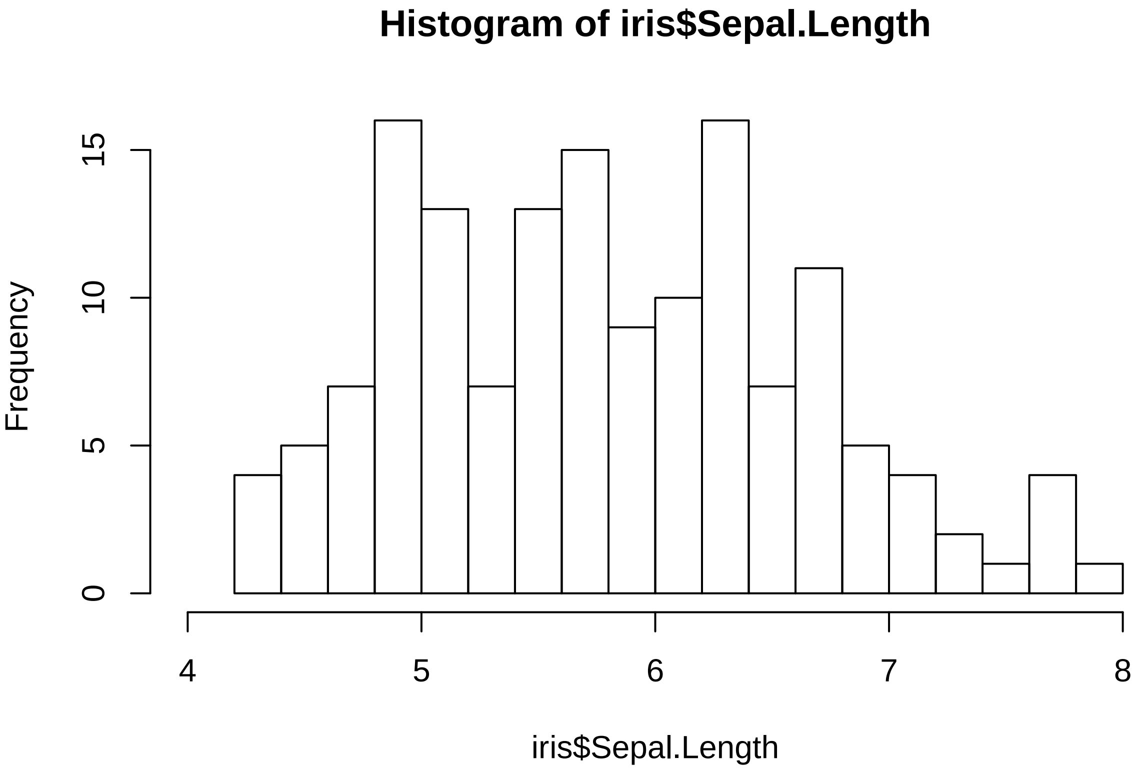
Let's give the histogram more detail by increasing the number of bins. This is done with the *breaks* argument:

```
hist(iris$Sepal.Length, breaks = 15)
```



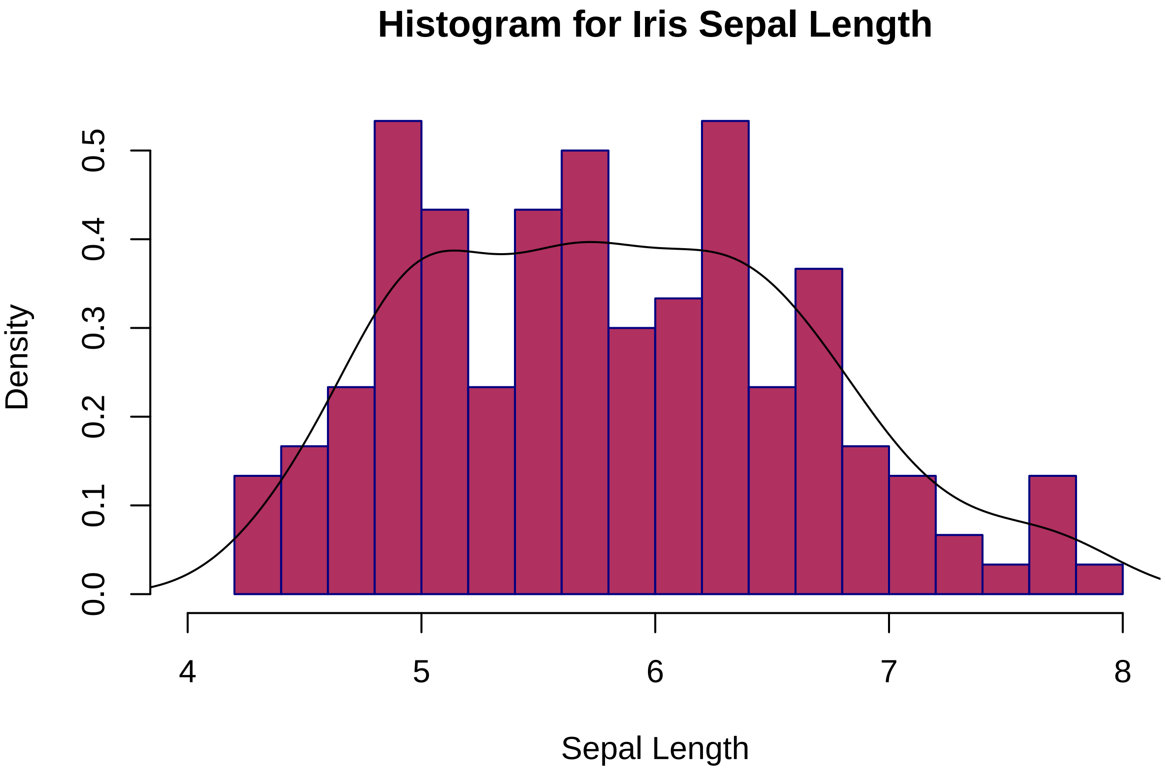
Hmm. . . the x axis is a little off. Let's adjust it with the *xlim* argument:

```
hist(iris$Sepal.Length, breaks = 25, xlim = c(4, 8))
```



The are many other options available for the *hist()* function. Using **?hist** or **help(hist)**, see what else you can find that will improve the figure and type your command in the field below (e.g., how about changing the x axis title?).

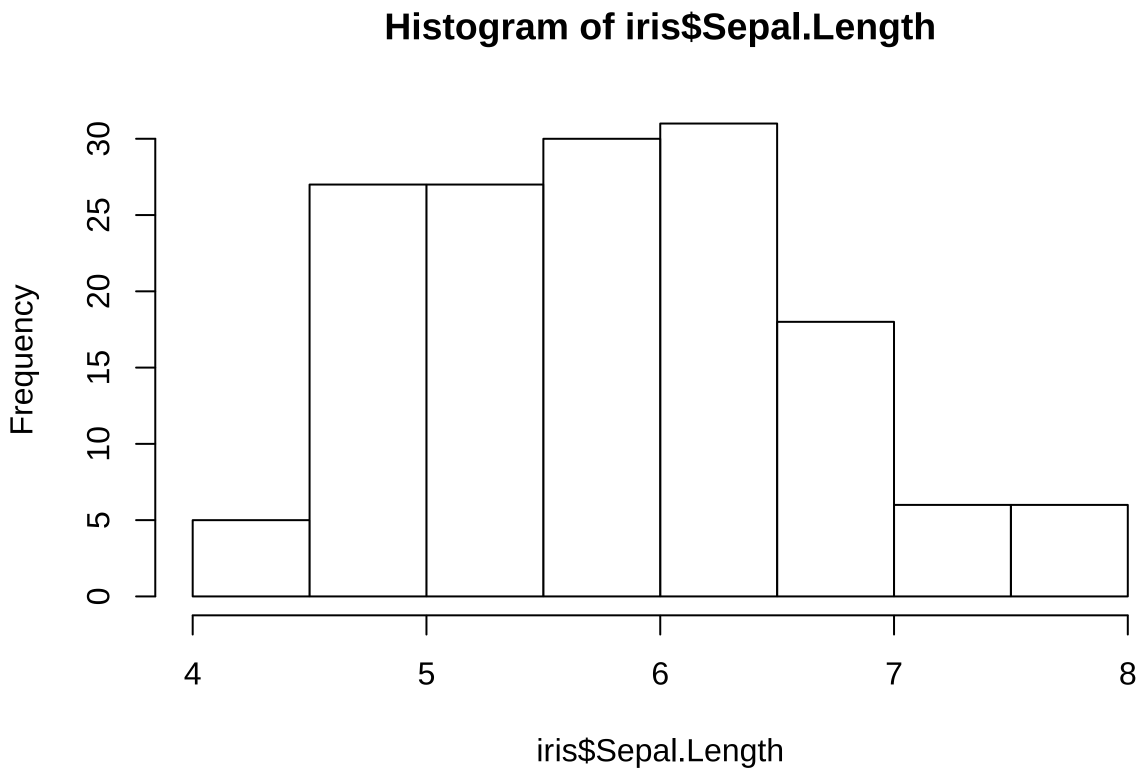
```
# See https://www.r-bloggers.com/how-to-make-a-histogram-with-basic-r/ for more examples
hist(iris$Sepal.Length, breaks = 15, xlim = c(4, 8), border="navy", col="maroon",
     xlab="Sepal Length", main="Histogram for Iris Sepal Length", prob = TRUE)

lines(density(iris$Sepal.Length))
```



By default, *hist()* shows the frequency of occurrence of values, as indicated by the y-axis label. If you set 'prob=TRUE', *hist()* will calculate the **likelihood of occurrance** of the value interval on the x-axis, i.e., a probability density. The *lines()* function adds a density curve to the plot.

On a side note…remember when we introduced **lists** earlier? A common use of lists is to gather different types of R objects in one container. Many functions, including *hist()*, return a list of different values. Each attribute of the returned histogram correlates to a specific component of the list. For example, if we assign the output of *hist(iris$Sepal.Length)* to a variable, we can capture the returned list:

```
sepal_hist <- hist(iris$Sepal.Length)
```



```
sepal_hist
```

```
## $breaks
## [1] 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
##
## $counts
## [1]  5 27 27 30 31 18  6  6
##
## $density
## [1] 0.06666667 0.36000000 0.36000000 0.40000000 0.41333333 0.24000000
## [7] 0.08000000 0.08000000
##
## $mids
## [1] 4.25 4.75 5.25 5.75 6.25 6.75 7.25 7.75
##
## $xname
## [1] "iris$Sepal.Length"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Take a look at '?hist' for descriptions of some of these components.

There are many other built-in data sets in R. To see a list…

```
data()
```

You can also download .csv and .doc file versions here: *https://vincentarelbundock.github.io/Rdatasets/datasets.html*. We look at an example of this in the next section.

Loading external data

So far we've been working with the *iris* data set built into R. What if we want to load a .csv file from another source? To demonstrate this, we'll use the *iris* data set again, but this time we'll load it from an external source using the *read.csv()* function.

```
iris_df <-read.csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv",
                  header = TRUE, sep=",", strip.white=TRUE, stringsAsFactors=FALSE)
```

As you can see, there are several additional arguments given. *strip.white=TRUE* removes white space from the beginning and end of unquoted values in a character field (numeric values are always trimmed). This argument should be used in conjunction with the *sep* argument. *stringsAsFactors=FALSE* forces R to set the data type of strings to 'character'. You can check this using the *str()* function we introduced ealier:

```
str(iris_df)
```

```
## 'data.frame':    150 obs. of  6 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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     : chr  "setosa" "setosa" "setosa" "setosa" ...
```

Importing data from other software

With R, you can import, process, and analyze data in many other file formats. Below we will show examples of libraries that are used to read SAS, Stata, SPSS, and Excel files. These libraries as well as others are introduced in the Data Camp courses **"Importing Data in R (Part 1)"** (*https://www.datacamp.com/courses/importing-data-in-r-part-2*) and **Part 2** (*https://www.datacamp.com/courses/importing-data-in-r-part-1*).

Load SAS, Stata, and SPSS files with haven

With the haven package, R has the ability to read and work with files generated from other statistical analysis packages. More detail on this function at *https://rpubs.com/potentialwjy/ImportDataIntoR04*.

```
# install the haven package
#install.packages("haven")

# Load the haven package
library(haven)

# Another option is the 'foreign' library
```

> — You can also install packages by going to **Tools** –> **Install Packages** on the R Studio menu bar. A window will pop up that allows you to choose the repository and type the name of the library you are looking for. An idexed list of available names should be revealed as you are typing.
> —

Loading SAS files:

```
# Load the haven package
library(haven)

# Import sales.sas7bdat
sales <- read_sas("http://assets.datacamp.com/production/course_1478/datasets/sales.sas7bdat")

# Display 'sales' structure and a summary
str(sales)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    431 obs. of  4 variables:
##  $ purchase: num  0 0 1 1 0 0 0 0 0 0 ...
##  $ age     : num  41 47 41 39 32 32 33 45 43 40 ...
##  $ gender  : chr  "Female" "Female" "Female" "Female" ...
##  $ income  : chr  "Low" "Low" "Low" "Low" ...
##  - attr(*, "label")= chr "SALES"
```

```
summary(sales)
```

```
##     purchase           age           gender             income         
##  Min.   :0.0000   Min.   :23.00   Length:431         Length:431        
##  1st Qu.:0.0000   1st Qu.:35.00   Class :character   Class :character  
##  Median :0.0000   Median :39.00   Mode  :character   Mode  :character  
##  Mean   :0.3759   Mean   :38.97                                        
##  3rd Qu.:1.0000   3rd Qu.:43.00                                        
##  Max.   :1.0000   Max.   :58.00
```

```
# That summary doesn't look right, so let's convert the appropriate variables to factors
sales$purchase <- factor(sales$purchase)
sales$gender <- factor(sales$gender)
sales$income <- factor(sales$income)

# Display 'sales' structure and a summary again and see the difference
str(sales)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    431 obs. of  4 variables:
##  $ purchase: Factor w/ 2 levels "0","1": 1 1 2 2 1 1 1 1 1 1 ...
##  $ age     : num  41 47 41 39 32 32 33 45 43 40 ...
##  $ gender  : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ income  : Factor w/ 3 levels "High","Low","Medium": 2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "label")= chr "SALES"
```

```
summary(sales)
```

```
##  purchase      age           gender       income   
##  0:269    Min.   :23.00   Female:240   High  :155  
##  1:162    1st Qu.:35.00   Male  :191   Low   :132  
##           Median :39.00                Medium:144  
##           Mean   :38.97                            
##           3rd Qu.:43.00                            
##           Max.   :58.00
```

```
# Where am I?
getwd()
```

```
## [1] "/Users/mbc400/Box Sync/GitHub/Intro_to_R"
```

```r
# Export contents of the SAS file as .csv
write.csv(sales, file = "Data_Files/sales_sas.csv", row.names = FALSE)
```

Loading Stata files:

```r
# Load the haven package
library(haven)

# Import the data from the URL
sugar <- read_dta("http://assets.datacamp.com/production/course_1478/datasets/trade.dta")

# Structure of sugar
str(sugar)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  5 variables:
##  $ Date    :Class 'labelled'  atomic [1:10] 10 9 8 7 6 5 4 3 2 1
##   .. ..- attr(*, "label")= chr "Date"
##   .. ..- attr(*, "format.stata")= chr "%9.0g"
##   .. ..- attr(*, "labels")= Named num [1:10] 1 2 3 4 5 6 7 8 9 10
##   .. .. ..- attr(*, "names")= chr [1:10] "2004-12-31" "2005-12-31" "2006-12-31" "2007-12-31" ...
##  $ Import  : atomic  37664782 16316512 11082246 35677943 9879878 ...
##   ..- attr(*, "label")= chr "Import"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Weight_I: atomic  54029106 21584365 14526089 55034932 14806865 ...
##   ..- attr(*, "label")= chr "Weight_I"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Export  : atomic  5.45e+07 1.03e+08 3.79e+07 4.85e+07 7.15e+07 ...
##   ..- attr(*, "label")= chr "Export"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Weight_E: atomic  9.34e+07 1.58e+08 8.80e+07 1.12e+08 1.32e+08 ...
##   ..- attr(*, "label")= chr "Weight_E"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  - attr(*, "label")= chr "Written by R."
```

```r
# Convert values in Date column to dates
sugar$Date <- as.Date(as_factor(sugar$Date))
```

```
## Warning in strptime(xx, f <- "%Y-%m-%d", tz = "GMT"): unknown timezone
## 'zone/tz/2018c.1.0/zoneinfo/America/Chicago'
```

```r
# Check the structure again; the data type of 'Date' should be changed
str(sugar)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  5 variables:
##  $ Date    : Date, format: "2013-12-31" "2012-12-31" ...
##  $ Import  : atomic  37664782 16316512 11082246 35677943 9879878 ...
##   ..- attr(*, "label")= chr "Import"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Weight_I: atomic  54029106 21584365 14526089 55034932 14806865 ...
##   ..- attr(*, "label")= chr "Weight_I"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Export  : atomic  5.45e+07 1.03e+08 3.79e+07 4.85e+07 7.15e+07 ...
##   ..- attr(*, "label")= chr "Export"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  $ Weight_E: atomic  9.34e+07 1.58e+08 8.80e+07 1.12e+08 1.32e+08 ...
##   ..- attr(*, "label")= chr "Weight_E"
##   ..- attr(*, "format.stata")= chr "%9.0g"
##  - attr(*, "label")= chr "Written by R."
```

Loading SPSS files

```
# Load the haven package
library(haven)

# Import person.sav
traits <- read_sav("http://assets.datacamp.com/production/course_1478/datasets/person.sav")

# Summarize traits
summary(traits)
```

```
##     Neurotic      Extroversion   Agreeableness   Conscientiousness
##  Min.   : 0.00   Min.   : 5.00   Min.   :15.00   Min.   : 7.00
##  1st Qu.:18.00   1st Qu.:26.00   1st Qu.:39.00   1st Qu.:25.00
##  Median :24.00   Median :31.00   Median :45.00   Median :30.00
##  Mean   :23.63   Mean   :30.23   Mean   :44.55   Mean   :30.85
##  3rd Qu.:29.00   3rd Qu.:34.00   3rd Qu.:50.00   3rd Qu.:36.00
##  Max.   :44.00   Max.   :65.00   Max.   :73.00   Max.   :58.00
##  NA's   :14      NA's   :16      NA's   :19      NA's   :14
```

```
# Print out a subset
subset(traits, Extroversion > 40 & Agreeableness > 40)
```

```
## # A tibble: 8 x 4
##   Neurotic Extroversion Agreeableness Conscientiousness
##      <dbl>        <dbl>         <dbl>             <dbl>
## 1       38           43            49                29
## 2       20           42            46                31
## 3       18           42            49                31
## 4       42           43            44                29
## 5       30           42            51                24
## 6       18           42            50                25
## 7       27           45            55                23
## 8       18           43            57                34
```

The above command prints out a what is called a **tibble**. This is an improvement on the standard data frame in that it provides a nicer printing method than the original. When data is loaded into a tibble, R does not automatically convert strings to factors, changes the names of variables, or create row names. The data type of each column is displayed in the second row for convenience. More on tibbles can be found here: *https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html*

Loading Excel Files

For this next exercise, we'll introduce a few commonly-used functions for navigating through the directory structure on your computer through R. *getwd()*, *dir()*, and *setwd()* allow you to identify your current directory, print the contents, and change to another directory, respectively:

```
# Where am I?
getwd()
```

```
## [1] "/Users/mbc400/Box Sync/GitHub/Intro_to_R"
```

```
# Change my working directory to './Data_Files'
setwd("./Data_Files")

# List the contents of the directory
dir()
```

```
## [1] "FSIS-Recall-Summary-2014.xlsx" "sales_sas.csv"
```

We see that our current folder contains the .xlsx file of interest. Let's install and load the **readxl** package, add our newly-learned *setwd()* command to move to the right directory, then use the *excel_sheets()* and *read_excel()* functions to check out the Excel file contents:

```
# Other libraries for reading Excel data include gdata (read.xls) and the XLConnect package
#
# Install the library if necessary
# installed.packages("readxl")

# Load the library
library(readxl)

# Where am I?
getwd()
```

```
## [1] "/Users/mbc400/Box Sync/GitHub/Intro_to_R"
```

```
# Change my working directory
setwd("./Data_Files")

# Show the sheets in this .xlsx file
# Data file source: https://catalog.data.gov/dataset/summary-of-recall-cases-in-calendar-year
excel_sheets("FSIS-Recall-Summary-2014.xlsx")
```

```
## [1] "Sheet1" "Sheet2"
```

```
# Just print out sheet 1
read_excel("FSIS-Recall-Summary-2014.xlsx", sheet = 1)
```

```
## # A tibble: 95 x 6
##    `CY 2014 Recalls`          X__1         X__2
##                <chr>         <chr>        <chr>
##  1       Recall Date Recall Number Recall Class
##  2             41649      001-2014            I
##  3             41652      002-2014            I
##  4             41654      003-2014            I
##  5             41656      004-2014           II
##  6             41656      005-2014            I
##  7             41658      006-2014           II
##  8             41666      007-2014           II
##  9             41669      008-2014            I
## 10             41671      009-2014           II
## # ... with 85 more rows, and 3 more variables: X__3 <chr>, X__4 <chr>,
## #   X__5 <chr>
```

```
# Store sheet 1 from the recall file in a list
recall_list <- read_excel("FSIS-Recall-Summary-2014.xlsx", sheet = 1, skip = 1,
                          col_types = c("date", "text", "guess", "text", "text", "numeric"))
```

```r
# Check out the structure of recall_list
str(recall_list)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    94 obs. of  6 variables:
##  $ Recall Date      : POSIXct, format: "2014-01-10" "2014-01-13" ...
##  $ Recall Number    : chr  "001-2014" "002-2014" "003-2014" "004-2014" ...
##  $ Recall Class     : chr  "I" "I" "I" "II" ...
##  $ Product          : chr  "Mechanically Separated Chicken Products" "Various Beef Products" "Beef F
##  $ Reason for Recall: chr  "Salmonella" "Other" "Undeclared Allergen" "Undeclared Allergen" ...
##  $ Pounds Recalled  : num  33840 42103 2664 130000 67113 ...
```

```r
# Change object type for "Reason for Recall" column to 'factor'
recall_list$"Reason for Recall" <- as.factor(recall_list$"Reason for Recall")

# What are the most common causes of recalls in this data set?
# Using the 'sort()' and 'table()' functions...
sort(table(recall_list$"Reason for Recall"), decreasing = TRUE)
```

```
## 
##                      Undeclared Allergen 
##                                       43 
##                                    Other 
##                                       23 
##                   Listeria monocytogenes 
##                                        7 
##                      Extraneous Material 
##                                        6 
##                          E. coli O157:H7 
##                                        4 
##                        Processing Defect 
##                                        4 
##                               Salmonella 
##                                        4 
##                     Undeclared Substance 
##                                        2 
## E. coli O103, O111, O121, O145, O26, O45 
##                                        1
```

```r
# You could also just call 'summary'...
sort(summary(recall_list$"Reason for Recall"), decreasing = TRUE)
```

```
##                      Undeclared Allergen 
##                                       43 
##                                    Other 
##                                       23 
##                   Listeria monocytogenes 
##                                        7 
##                      Extraneous Material 
##                                        6 
##                          E. coli O157:H7 
##                                        4 
##                        Processing Defect 
##                                        4 
##                               Salmonella 
##                                        4
##                       Undeclared Substance
##                                          2
## E. coli O103, O111, O121, O145, O26, O45
##                                          1
```

In Conclusion. . .

What we've gone over in this introduction only scratches the surface of what can be done in R. To become a proficient user of R, you will need to practice coding. Remember that you now have acess to the Data Camp platform (*https://www.datacamp.com/*) for the summer session (thanks to Christina Maimone at NUIT Research Computing). The Data Camp R courses provide you with self-paced, hands-on instruction and practice. You should have received a Data Camp invitation email. **You will need to create an account on the Data Camp site with your university email to access the online content.**

If you are at the beginner level with R and would like to complete a learning track in Data Camp, the "Data Analyst with R" track is recommended (*https://www.datacamp.com/tracks/data-analyst-with-r*). For those with more time and interest, the "Data Scientist with R" career track (*https://www.datacamp.com/tracks/data-scientist-with-r*) is an option (this is approximately 95 hours of instruction). Other tracks are available too, so feel free to look around and see what interests you.

In addition to the Data Camp site itself, **you also have access to material Christina has uploaded to Canvas**. This includes references, discussion boards, and exercises for you to try: *https://canvas.northwestern.edu/courses/52737*

If you would like to continue to have access to Data Camp next Fall, you will have to apply again when the time comes. Check this link for updated information as we approach the Fall quarter: *http://www.it.northwestern.edu/research/campus-events/data-camp.html*

References and Tutorials for R:

1. Data Camp: *https://www.datacamp.com/courses/tech:r*
2. About R: *https://www.r-project.org/about.html*
3. R Help: *https://www.r-project.org/help.html*
4. R Bloggers: *https://www.r-bloggers.com/*
5. Cross Validated: *https://stats.stackexchange.com/*
6. The Art of R Programming (Book): *https://www.amazon.com/Art-Programming-Statistical-Software-Design/dp/1593273843*
7. R for Data Science (Book): *https://www.amazon.com/Data-Science-Transform-Visualize-Model/dp/1491910399*
8. R Cookbook (Book): *https://www.amazon.com/Cookbook-Analysis-Statistics-Graphics-Cookbooks/dp/0596809158*
9. R Markdown reference guide: *https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf*
10. R Markdown cheat sheet: *https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf*
11. *https://joelkuiper.eu/R-workshop*
12. *http://www.r-tutor.com/r-introduction/data-frame/data-import*