Contents

Introduction:

Hi everybody, my name is Connor and I will be helping you get through 421!

We're going to start with a walkthrough of R.

Lesson 0: Basics

In general:

- We will be working with what is called a *script*. This is similar to a do file in Stata. It's basically your workspace.
- To execute a script, hit cmd+return (control+enter if on PC.) To save the script, cmd+s. There are other shortcuts as well. If you want to run a specific line, then you can move your cursor to that line and hit cmd+return and the R script will only run that one line.
- R uses *functions*, which we apply to *objects*. More on this shortly, but if you aren't sure what a function does, or how it works, you can use ? before the function to get the documentation. Ex:

```
?mean
```

As a note, these grey boxes will be where I am typing my code for you to reference.

There are a ton of different types of objects (numeric (numbers), character (letters) and logical (true false statements) are the most common types), and not all functions will work on all objects. Let's talk a bit about objects.

Lesson 1: All things are objects

An object is an assignment between a name and a value. You assign values to names using `<-` or `=`. The first assignment symbol consists of a `<` next to a dash `-` to make it look like an arrow.

If we want to make an object name 'a' refer to the number '2', we can do that by:

```
a <- 2
# a = 2
a
```

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

Note: The # comments out code meaning R will not evaluate the code. This is a convenient way to write yourself notes as you're coding.

You can combine objects together as well, assigning

```
#assign the value of 3 to the name b
b <- 3
#assign the value of b (3) plus the value of a (2), to a new name, c.
c <- a + b
#display c
c
```

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

When you wrap parentheses around an assignment, R will both (1) assign the value to the name and (2) print out the value (to the screen) that you've assigned.

```
#let's print two lines. Parentheses will print this
(d <- c * 3)
```

```
## [1] 15
```

```
#R-markdown, what this is written in, will automatically output the last line of a cell.
d + 3 - 1 + c
```

```
## [1] 22
```

Objects can also contain more than one value. What do you think the object 1:10 does?

```
(tmp <- 1:10)
```

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

This displays a sequence of integers (whole numbers), going from the first value to the last one, ie, 9:10 will return 9 10.

One common way (illustrated above) to store multiple values is via a `vector`.

Lesson 2: Functions

Functions are operations you can apply to your created object.

Examples: `head`, `tail`, `mean`, `median`, `sd`, `summary`, `min`, `max`

These functions are good at summarizing data in a variety of ways. Let's see how they work

```
#print the first few objects in 'tmp'
head(tmp)
```

```
## [1] 1 2 3 4 5 6
```

```
#print the first 3 objects in 'tmp'
head(tmp, 3)
```

```
## [1] 1 2 3
```

```
#print the last few objects in 'tmp'
tail(tmp)
```

```
## [1]  5  6  7  8  9 10
```

```r
#mean of our vector tmp
mean(tmp)
```

```
## [1] 5.5
```

```r
#median of our vector
median(tmp)
```

```
## [1] 5.5
```

```r
#standard deviation of our vector
sd(tmp)
```

```
## [1] 3.02765
```

```r
#*IMPORTANT* Print a summary of our object. This can work on many object types and is useful to get an
summary(tmp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

```r
#Print minimum of the vector
min(tmp)
```

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

```r
#print max of a vector
max(tmp)
```

```
## [1] 10
```

One caveat: what do you think will happen when I run the following?

```r
#Let's make a bad vector
bad <- c(tmp, "red", "dinosaurs", NULL)
(tail(bad, 4))
```

```
## [1] "9"         "10"        "red"       "dinosaurs"
```

```r
(tail(tmp, 4))
```

```
## [1]  7  8  9 10
```

Aside from the last three objects, do these two sets look different to you? We'll get back to this.

Mathematical operators

We can also use R to do anything you could do in a calculator, that is, multiply, divide, and so forth. Most of these are self explanatory

```r
#add
11 + 2
```

```
## [1] 13
```

```r
#subtract
11 - 2
```

```
## [1] 9
```

```r
#multiply
11 * 2
```

```
## [1] 22
```

```
#divide
11 / 2
```

```
## [1] 5.5
```

```
#Find remainder (this is surprisingly useful)
(11 %% 2)
```

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

```
#Find integer division, ie, how many times does 2 fit into 11 as a whole number
11 %/% 2
```

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

```
#Power
11 ^ 2
```

```
## [1] 121
```

Functions create objects, e.g., `c()` creates vectors of values for individual objects' values.

```
vec_a <- c(1, 5, 8, 20)
vec_a
```

```
## [1]  1  5  8 20
```

You can also apply mathematical operators to vectors.

```
vec_a * 2
```

```
## [1]  2 10 16 40
```

```
vec_a ^ 2
```

```
## [1]   1  25  64 400
```

```
vec_a - vec_a
```

```
## [1] 0 0 0 0
```

Finally, keep track of missing values (`NA`) and infinity (`Inf`)!

```
# Division by NA
vec_a / NA
```

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

```
# Division by zero. This creates a special value in R called 'inf'
vec_a / 0
```

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

Lesson 2: Types of Objects

So far, you've seen `numeric` objects (which can be `numeric` or `integer`). We can see what kind of class an object is by using the `class()` function.

```
(class(a))
```

```
## [1] "numeric"
```

```
(class(tmp))
```

```
## [1] "integer"
```

```
(class(vec_a))
```

```
## [1] "numeric"
```

Another common class of objects is `character`:

```
#Let's create a character object. These are surrounded by either "" or ''. This distinguishes them from
(some_text <- "I have a birthday, but it is not today.")
```

```
## [1] "I have a birthday, but it is not today."
```

```
class(some_text)
```

```
## [1] "character"
```

Lastly, we have `logical` objects.

Logical objects are essentially anything you could classify as a true/false statement, like, the sky is purple = FALSE. Caustic is the best legend = FALSE.

```
#These generally use the characters less than (<), greater than (>), or is equivalent (==)
(2 < 3)
```

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

```
(2 > 3)
```

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

```
(2 == 3)
```

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

```
(TRUE == TRUE)
```

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

```
class(TRUE)
```

```
## [1] "logical"
```

```
#We can also do this with vectors. This will compare each element in the vector to check your provided
c(1,2,3,4,5) < c(2,0,1,2,3)
```

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

which have some special operators

```
# AND
TRUE & FALSE
```

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

```
# OR
TRUE | FALSE
```

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

Be careful. Vectors only hold 1 type of object, let's look at our vector 'bad' again.

```
class(bad)
```

```
## [1] "character"
```

Why is it a character now? Well, R is trying to help us. By adding some words to the end of the vector, R will automatically transform the entire vector to a character class. We can however, store multiple types of values by using special data types.

In particular, you guys will be working with data frames and tibbles. We can create a data frame with the `data.frame()` function, passing it three *arguments* that are *vectors.* Let's see how that works:

```
our_df <- data.frame(
  v1 = 1:4,
  v2 = c(T, T, F, T),
  v3 = c("a", "b", "c", "d")
)
our_df
```

```
##   v1    v2 v3
## 1  1  TRUE  a
## 2  2  TRUE  b
## 3  3 FALSE  c
## 4  4  TRUE  d
```

You can think about data frames (`data.frame`) as spreadsheets, each column as a new variable, and each row has a new observation. We'll generally read data into R as a data frame.

Lesson 3: Load Packages

Base R (what comes installed on your computer) is an incredibly powerful programming language, but one of the best features of R are its packages, which are remotely stored functions written by anybody. You could even write a package if you wanted! This open source nature allows R to be extremely flexible. For now, we will load the `pacman` package management package, and then the `ISLR` package which has a number of datasets.

Let's start by loading packages. Uncomment the install.packages function to get the pacman package to install. If you already have some of these packages, feel free to delete lines. The install.packages function can take a vector of package names, as characters, to install all of the above.

```
install.packages(c("pacman","ISLR", "ggplot2"), dependencies=T, repos = "http://cran.us.r-project.org")
```

```
## Installing packages into '/Users/connor/Library/R/3.5/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/41/9t4d_k797tddkd5yfnp9v3y40000gn/T//Rtmp4d31ZF/downloaded_packages
```

```
#pacman will allow us to load packages intelligently. Load it with the library() function
library(pacman)
#p_load is pacman's 'library' and features a number of improvements. Load next two packages.
p_load(ISLR, ggplot2)
```

You can also do all of this in base R, but it's not as efficient.

```
#the following code does the same thing. Not as good. More packages you have, the more convinient pacma
install.packages(c("ISLR","ggplot2"),dependencies = T, repos="http://cran.us.r-project.org")
```

```
## Installing packages into '/Users/connor/Library/R/3.5/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/41/9t4d_k797tddkd5yfnp9v3y40000gn/T//Rtmp4d31ZF/downloaded_packages
```

```
library(ISLR)
library(ggplot2)
```

Lesson 4: More about data frames

We installed and loaded `ISLR` because it has a nice dataset for learning about data frames: `Auto` (you may have seen this one before...).

Let's spend some time familiarizing ourselves with the dataset. We can use many of the familiar tools we had before, namely `head()`, `tail()`, and `summary()`.

If we want to look at a specific column of a dataframe, we can do this by writing `dataframe$columnName`

```
#I always like to get a summary of my data before doing much of anything
(summary(Auto))
```

```
##       mpg          cylinders      displacement     horsepower
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0
##
##      weight      acceleration        year           origin
##  Min.   :1613   Min.   : 8.00   Min.   :70.00   Min.   :1.000
##  1st Qu.:2225   1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000
##  Median :2804   Median :15.50   Median :76.00   Median :1.000
##  Mean   :2978   Mean   :15.54   Mean   :75.98   Mean   :1.577
##  3rd Qu.:3615   3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000
##  Max.   :5140   Max.   :24.80   Max.   :82.00   Max.   :3.000
##
##                  name
##  amc matador       :  5
##  ford pinto        :  5
##  toyota corolla    :  5
##  amc gremlin       :  4
##  amc hornet        :  4
##  chevrolet chevette:  4
##  (Other)           :365
```

```
#Let's call a specific column
(Auto$cylinders)
```

```
##   [1] 8 8 8 8 8 8 8 8 8 8 8 8 8 8 4 6 6 6 4 4 4 4 4 4 6 8 8 8 8 4 4 4 6 6 6
##  [36] 6 6 8 8 8 8 8 8 8 8 6 4 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 8 8 8 8 8 8 8 8
##  [71] 3 8 8 8 8 4 4 4 4 4 4 4 4 4 4 8 8 8 8 8 8 8 8 8 8 8 8 6 6 6 6 6 4 8 8 8
## [106] 8 6 4 4 4 3 4 6 4 8 8 4 4 4 4 8 4 6 8 6 6 6 4 4 4 4 6 6 6 8 8 8 8 8 4
## [141] 4 4 4 4 4 4 4 4 4 4 6 6 6 6 8 8 8 8 6 6 6 6 6 8 8 4 4 6 4 4 4 4 6 4 6
## [176] 4 4 4 4 4 4 4 4 4 4 8 8 8 8 6 6 6 6 4 4 4 4 6 6 6 6 4 4 4 4 4 8 4 6 6
## [211] 8 8 8 8 4 4 4 4 4 8 8 8 8 6 6 6 6 8 8 8 8 4 4 4 4 4 4 4 4 6 4 3 4 4 4
```

```
## [246] 4 4 8 8 8 6 6 6 4 6 6 6 6 6 6 8 6 8 8 4 4 4 4 4 4 4 4 5 6 4 6 4 4 6 6
## [281] 4 6 6 8 8 8 8 8 8 8 8 4 4 4 4 5 8 4 8 4 4 4 4 4 6 6 4 4 4 4 4 4 4 4 6
## [316] 4 4 4 4 4 4 4 4 4 4 5 4 4 4 4 6 3 4 4 4 4 4 6 4 4 4 4 4 4 4 4 4 4 4 4
## [351] 4 4 4 4 4 6 6 6 6 8 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6 4 6 4
## [386] 4 4 4 4 4 4 4
```

```
#Look at the first few rows of our dataframe
(head(Auto))
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```
#look at the last few rows of our dataframe
(tail(Auto))
```

```
##     mpg cylinders displacement horsepower weight acceleration year origin
## 392  27         4          151         90   2950         17.3   82      1
## 393  27         4          140         86   2790         15.6   82      1
## 394  44         4           97         52   2130         24.6   82      2
## 395  32         4          135         84   2295         11.6   82      1
## 396  28         4          120         79   2625         18.6   82      1
## 397  31         4          119         82   2720         19.4   82      1
##                 name
## 392 chevrolet camaro
## 393  ford mustang gl
## 394        vw pickup
## 395    dodge rampage
## 396      ford ranger
## 397       chevy s-10
```

Indexing

We can also call columns and rows by their numeric 'index.' This is a numerical value that R holds onto as a reference point. As a note for computer-saavy readers: r begins indexing at 1. Let's see what this looks like:

```
#Here's our first column
Auto$cylinders
```

```
##   [1] 8 8 8 8 8 8 8 8 8 8 8 8 8 4 6 6 6 4 4 4 4 4 4 6 8 8 8 8 4 4 4 6 6 6
##  [36] 6 6 8 8 8 8 8 8 8 8 6 4 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 8 8 8 8 8 8 8 8
##  [71] 3 8 8 8 8 4 4 4 4 4 4 4 4 4 4 8 8 8 8 8 8 8 8 8 8 8 6 6 6 6 6 4 8 8 8
## [106] 8 6 4 4 4 3 4 6 4 8 8 4 4 4 4 8 4 6 8 6 6 6 4 4 4 4 6 6 6 8 8 8 8 8 4
## [141] 4 4 4 4 4 4 4 4 4 4 6 6 6 6 8 8 8 8 6 6 6 6 6 8 8 4 4 6 4 4 4 4 6 4 6
## [176] 4 4 4 4 4 4 4 4 4 4 8 8 8 8 6 6 6 6 4 4 4 4 6 6 6 6 4 4 4 4 4 8 4 6 6
## [211] 8 8 8 8 4 4 4 4 4 8 8 8 8 6 6 6 6 8 8 8 8 4 4 4 4 4 4 4 4 6 4 3 4 4 4
## [246] 4 4 8 8 8 6 6 6 4 6 6 6 6 6 6 8 6 8 8 4 4 4 4 4 4 4 4 5 6 4 6 4 4 6 6
```

```
## [281] 4 6 6 8 8 8 8 8 8 8 8 4 4 4 4 5 8 4 8 4 4 4 4 4 6 6 4 4 4 4 4 4 4 4 6
## [316] 4 4 4 4 4 4 4 4 4 4 5 4 4 4 4 6 3 4 4 4 4 4 6 4 4 4 4 4 4 4 4 4 4 4 4
## [351] 4 4 4 4 4 6 6 6 6 8 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6 4 6 4
## [386] 4 4 4 4 4 4 4
```

```
#If we want the first element of that column, we can do that this way:
(Auto$cylinders[1])
```

```
## [1] 8
```

```
#we can also grab elements, say 1-10 in an array:
(Auto$cylinders[1:10])
```

```
##  [1] 8 8 8 8 8 8 8 8 8 8
```

```
#what about matrices?
(head(Auto))
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```
#row 2, column 1. just like matrix dimensions. row by col.
(Auto[2,1])
```

```
## [1] 15
```

```
#We can also grab slices of a dataframe, by passing the index a series of integers. Let's look at the s
Auto[2:3,3:4]
```

```
##   displacement horsepower
## 2          350        165
## 3          318        150
```

We can also use a few new summary functions to get some basic information out of our dataframe:

```
#Number of rows
(nrow(Auto))
```

```
## [1] 392
```

```
#Number of columns
(ncol(Auto))
```

```
## [1] 9
```

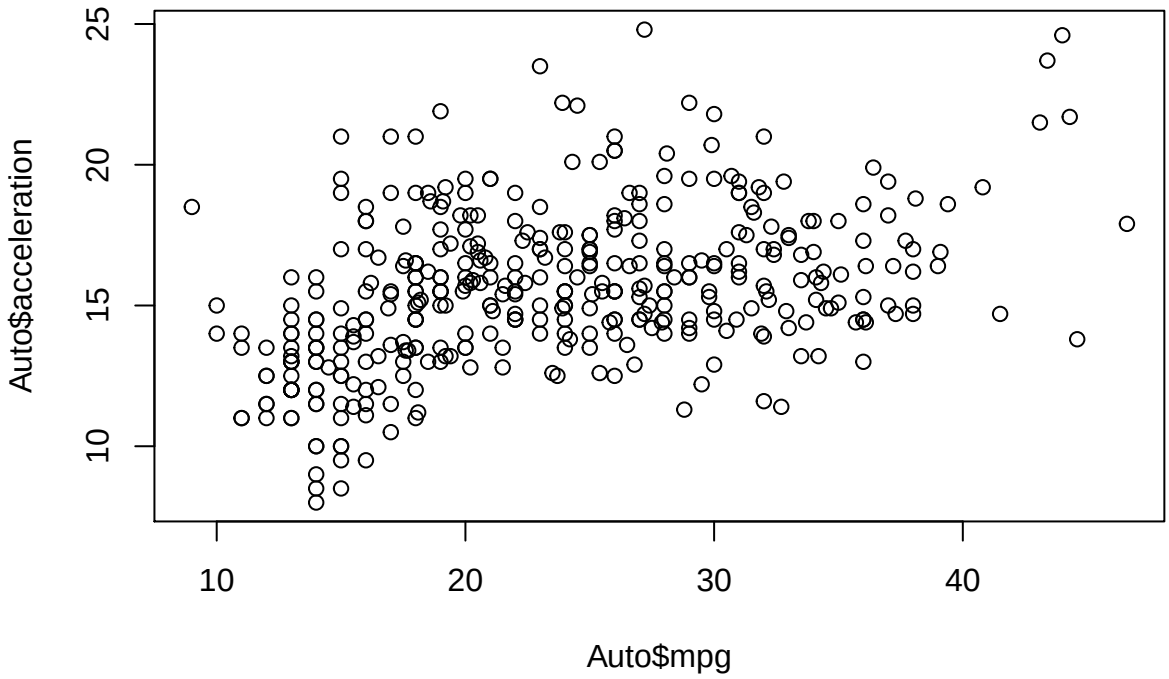
```
#Dimensions of the DF
dim(Auto)
```

```
## [1] 392   9
```

Lesson 5: Plotting

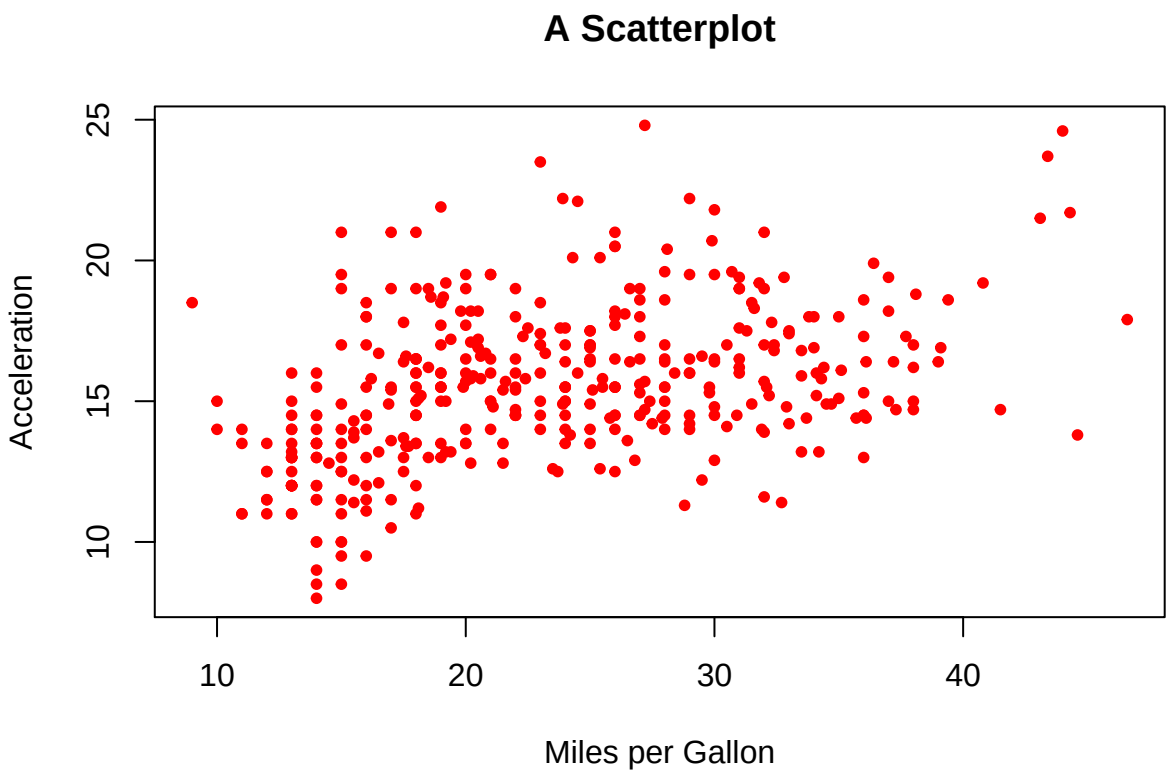
One of the most useful things in R is the myriad of ways it can visualize data in an appealing way. Let's dive in. The `plot(x,y)` function will produce a scatterplot,

```
#plot(x,y)
plot(Auto$mpg, Auto$acceleration)
```



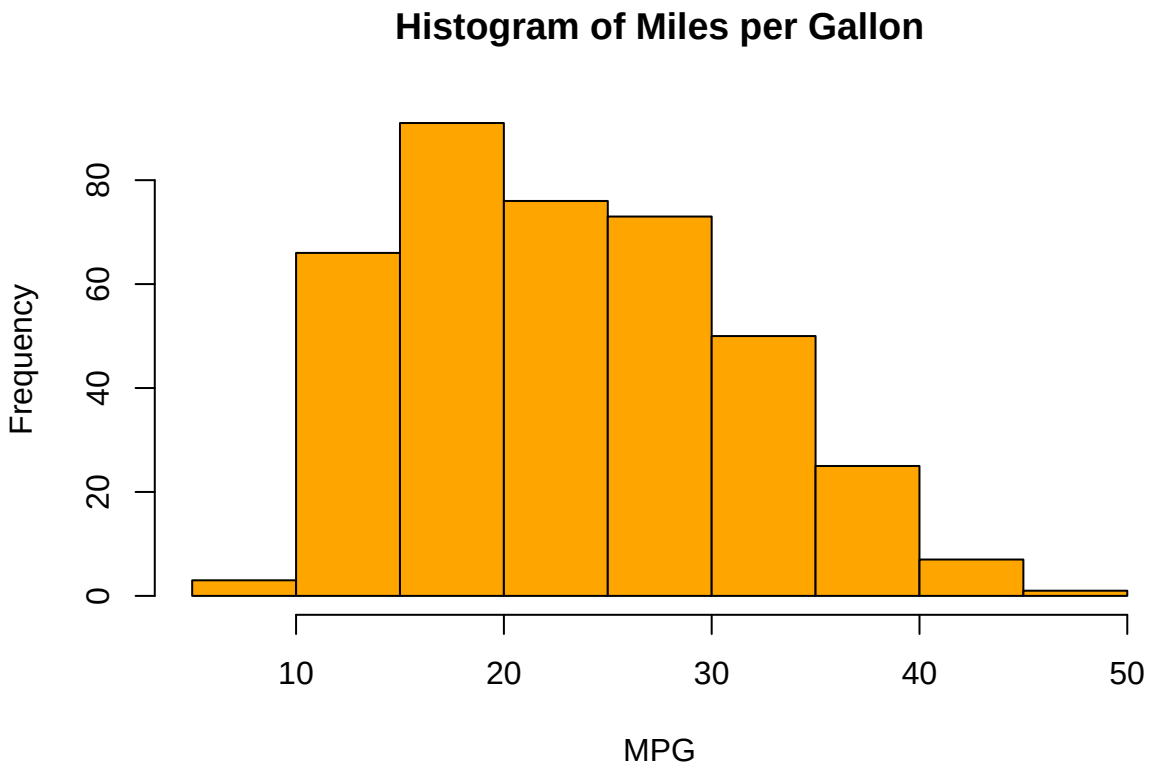
Okay, maybe I didn't start out on the best foot. That's not the prettiest thing in the world. Let's try again.

```
#First lets figure out what acceleration is: we can look at datasets if they have documentation
?Auto
#Now lets make our graph look a bit better
#Col will set a color, xlab will label the x-axis, ylab will label the y-axis, and pch is how plot chan
plot(Auto$mpg, Auto$acceleration, xlab="Miles per Gallon",
     ylab= "Acceleration", main="A Scatterplot",col="red",
     pch=20 )
```



We can also plot a histogram

```
hist(Auto$mpg,main= "Histogram of Miles per Gallon", xlab= "MPG", ylab="Frequency", col = "orange")
```



Lesson 6: Regression

What you guys came here for! Lots of regression. Regression is fairly straightforward if you are familiar with Stata.

We have a new function, `lm()`, and a new object type, called a `formula`. If we want to run a regression: $Y = \beta * X + \varepsilon$ we have to pass the lm function something it can use to know what to do. We do this with the `y ~ x1 + x2...` format.

```
#create an object w regression. lm for "linear model"
new_reg = lm(mpg ~ weight, data = Auto)
#and we can use the summary object to get a view very similar to Stata's
summary(new_reg)
```

```
##
## Call:
## lm(formula = mpg ~ weight, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9736  -2.7556  -0.3358   2.1379  16.5194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 46.216524   0.798673   57.87   <2e-16 ***
## weight      -0.007647   0.000258  -29.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.333 on 390 degrees of freedom
## Multiple R-squared:  0.6926, Adjusted R-squared:  0.6918
## F-statistic: 878.8 on 1 and 390 DF,  p-value: < 2.2e-16
```

Also, we can run regressions without defining a dataset.

```
#We don't NEED to pass a dataset, but we need to tell lm where to find the information. We can do that
summary(lm(Auto$mpg ~ Auto$weight))
```

```
##
## Call:
## lm(formula = Auto$mpg ~ Auto$weight)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9736  -2.7556  -0.3358   2.1379  16.5194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 46.216524   0.798673   57.87   <2e-16 ***
## Auto$weight -0.007647   0.000258  -29.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.333 on 390 degrees of freedom
## Multiple R-squared:  0.6926, Adjusted R-squared:  0.6918
## F-statistic: 878.8 on 1 and 390 DF,  p-value: < 2.2e-16
```

On your own, interpret these coefficients? What do these mean?

```
#Let's do another regression. This time, looking at the effect of origin on miles per gallon.
origin_reg <- lm(mpg ~ origin, data = Auto)
summary(origin_reg)
```

```
##
## Call:
## lm(formula = mpg ~ origin, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2416  -5.2533  -0.7651   3.8967  18.7115
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  14.8120     0.7164   20.68   <2e-16 ***
## origin        5.4765     0.4048   13.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.447 on 390 degrees of freedom
## Multiple R-squared:  0.3195, Adjusted R-squared:  0.3177
## F-statistic: 183.1 on 1 and 390 DF,  p-value: < 2.2e-16
```

Notice that `origin` variable is coded as numeric, but it really is a categorical varialbe (1 = American, 2 = European, 3 = Japanese). R doesn't know this, so it's treating it as if '3' is 3 times as origin-y as '1'. We don't want that.

```
#R is treating origin as a...
(class(Auto$origin))
```

```
## [1] "numeric"
```

```
#To visualize...
hist(Auto$origin,main= "Histogram of Origin", xlab= "Origin", ylab="frequency")
```

There's a way to fix this, by redefining `Auto$origin` using the `as.character()` function.

```
origin_reg <- lm(mpg ~ as.character(origin), data = Auto)
summary(origin_reg)
```

```
##
## Call:
## lm(formula = mpg ~ as.character(origin), data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.451  -5.034  -1.034   3.649  18.966
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            20.0335     0.4086  49.025   <2e-16 ***
## as.character(origin)2   7.5695     0.8767   8.634   <2e-16 ***
## as.character(origin)3  10.4172     0.8276  12.588   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.396 on 389 degrees of freedom
## Multiple R-squared:  0.3318, Adjusted R-squared:  0.3284
## F-statistic:  96.6 on 2 and 389 DF,  p-value: < 2.2e-16
```

When `lm()` sees a character value, it will automatically treat that variable as a dummy variable. Now, origin is equal to "1" rather than 1.

As an aside, and to test your memory: what is the interpretation of the intercept here?

If we wanted, we could also remove the intercept. This might give you a hint to the question above:

```r
#remove the intercept: we can do this by adding a '-1' into our formula.
origin_reg_no_int <- lm(mpg ~ as.character(origin)-1, data = Auto)
summary(origin_reg_no_int)
```

```
##
## Call:
## lm(formula = mpg ~ as.character(origin) - 1, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.451  -5.034  -1.034   3.649  18.966
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## as.character(origin)1  20.0335     0.4086   49.02   <2e-16 ***
## as.character(origin)2  27.6029     0.7757   35.59   <2e-16 ***
## as.character(origin)3  30.4506     0.7196   42.31   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.396 on 389 degrees of freedom
## Multiple R-squared:  0.9335, Adjusted R-squared:  0.933
## F-statistic:  1820 on 3 and 389 DF,  p-value: < 2.2e-16
```

And that's it! Welcome to R. Please feel free to play around with the datasets and get comfortable slicing dataframes, since you will likely be doing that often.

As practice, play around running regressions in the `ISLR` 'College' dataset. Run a regression on your own, and try to look at entire rows or entire columns to familiarize yourself with R.

Good luck, and I'll see you next week.