Introduction to R

Please remember to swipe in!

Lauren Cappiello, GradQuant Lead Consultant

September 12, 2018

Welcome to R

R is an open source software that offers tremendous flexibility and capability. Unfortunately some of this flexibility comes at the cost of a steeper learning curve, but have no fear!

What's so great about R?

- The open source nature means it is capable of almost anything (it just passed 10,000 packages!)
- Accessible and easy to download, no license restrictions
- Powerful enough for complex computing tasks, like machine learning
- Graphics capabilities surpass many other programs
- Works well with other programs like SQL, Excel, and Stata

Some disadvantages

- Steep learning curve
- Source documentation can be difficult to understand
- Open source means more packages, but also means potential errors – though if it's popular these tend to get solved quickly

Installing and Using R

Recommended software:

- R (https://www.r-project.org/)
- RStudio (https://www.rstudio.com/)
- RMarkdown:

```
install.packages("rmarkdown")
```

- R can be run without RStudio, but RStudio is a really nice GUI that will make it easier to work in R. (Note that R must be installed for RStudio to run.)
- RMarkdown isn't necessary for using R, but will allow you to write (and run) RCode within a document such as this one.

Comments

First, let's make R a little bit easier for everyone. When we use R, we have the option of adding comments to our code.

```
#This is a comment
```

Everything on a line to the left of a `#` is a comment. R will ignore comments, so you can use comments to give yourself and your collaborators an explanation; leave yourself a reminder; or skip a line of code.

Comments are worth your time!!

Mathematical Operations

Let's start with some simple mathematical operations.

```
2+2
```

```
## [1] 4
```

```
2^3
```

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

R uses standard order of operations

```
3+4*(4^2)
```

```
## [1] 67
```

Mathematical Operations

```
factorial(6)
```

```
## [1] 720
```

We've used our first function here. There's a lot to be said about functions, but for now we can see how to get help for a particular function:

```
?factorial
```

This will pull up the help file (RStudio) or open the help file in the default browser (R console).

Mathematical Operations

We can also test for (in)equalities

```
2 == 3
```

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

```
2 < 3
```

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

```
2^2 == 4
```

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

(Notice that we use "==" instead of "=" to test equality. The "=" operator is for setting things equal to one another.)

Mathematical Operations

That said, we need to be mindful of rounding error because of the way that R stores numbers. We expect $\sqrt{2}^2 = 2$, but…

```
sqrt(2)^2 == 2
```

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

To help, we can use the `all.equal` function. which tests "near equality".

```
all.equal(sqrt(2)^2, 2)
```

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

Data Storage and Manipulation

R can also store and use objects that we create, like the single value

```
x <- 2*3
```

which we can output using

```
print(x)
```

```
## [1] 6
```

or

```
x
```

```
## [1] 6
```

Data Storage and Manipulation

We can also create a vector of values using the `c` command, which combines all of the elements inside the parantheses into a vector or list.

```
y <- c(2,4,7)
y
```

```
## [1] 2 4 7
```

We can also access all or part of our vector:

```
y[1]
```

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

```
y[3]
```

```
## [1] 7
```

R numbers its vector and matrices starting at 1. If you're familiar with programming languages like C, this is something to be mindful of.

Data Storage and Manipulation

We can now work with an entire vector instead of individual values.

```
y+2
```

```
## [1] 4 6 9
```

```
x*y
```

```
## [1] 12 24 42
```

Aside

Aside: R works well with vector and matrix algebra, but that's not R's default!

```
z <- c(0.5,2,1,3,2.5,6)
y
```

```
## [1] 2 4 7
```

```
y*z
```

```
## [1]  1  8  7  6 10 42
```

This is not a dot- or a cross-product! Those are separate functions in R.

Data Storage and Manipulation

We can see a complete list of the objects we have saved

```
ls()
```

```
## [1] "x" "y" "z"
```

We can overwrite the objects we have saved

```
x <- 10
x <- 20
x
```

```
## [1] 20
```

Data Storage and Manipulation

We can also create sequences of variables

```
x <- 1:10
x
```

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

or

```
x <- seq(from = 1, to = 10, by = 1)
x
```

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

If we want to change the boundaries and increments,

```
x <- seq(from = 1, to = 12, by = .5)
x
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0
```

repeat values a certain number of times,

```
x <- rep(1, 10)
x
```

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

or repeat an entire vector a particular number of times

```
x <- rep(c(1,4), 10)
x
```

```
##  [1] 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
```

we can do those too!

17/59

Data Storage and Manipulation

Combining all of this…

```
rep(1:10, 10) + seq(1, 50.5, .5)
```

```
##   [1]  2.0  3.5  5.0  6.5  8.0  9.5 11.0 12.5 14.0 15.5  7.0  8.5 10.0 11.5
##  [15] 13.0 14.5 16.0 17.5 19.0 20.5 12.0 13.5 15.0 16.5 18.0 19.5 21.0 22.5
##  [29] 24.0 25.5 17.0 18.5 20.0 21.5 23.0 24.5 26.0 27.5 29.0 30.5 22.0 23.5
##  [43] 25.0 26.5 28.0 29.5 31.0 32.5 34.0 35.5 27.0 28.5 30.0 31.5 33.0 34.5
##  [57] 36.0 37.5 39.0 40.5 32.0 33.5 35.0 36.5 38.0 39.5 41.0 42.5 44.0 45.5
##  [71] 37.0 38.5 40.0 41.5 43.0 44.5 46.0 47.5 49.0 50.5 42.0 43.5 45.0 46.5
##  [85] 48.0 49.5 51.0 52.5 54.0 55.5 47.0 48.5 50.0 51.5 53.0 54.5 56.0 57.5
##  [99] 59.0 60.5
```

If need be, we can remove certain objects from the working environment

```
rm(x)
```

or clear the working environment entirely

```
rm(list=ls())
```

Matrices

R also has extensive capabilities in working with matrices.

```
mymatrix <- matrix(c(1,2,3,4,5,6), ncol=3, nrow=2)
mymatrix
```

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

We can get dimensions or take the transpose

```
dim(mymatrix)
```

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

```
t(mymatrix)
```

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

Matrices

We can access specific elements, rows, or columns.

```
mymatrix[2,3]
```

```
## [1] 6
```

```
mymatrix[2,]
```

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

```
mymatrix[,3]
```

```
## [1] 5 6
```

Numbers before the comma are row numbers and numbers after the comma refer to columns.

Matrices

We can find the inverse (of a square matrix).

```
mymatrix2 <- matrix(c(1,2,3,4,5,6,0,1,0), ncol=3, nrow=3)
solve(mymatrix2)
```

```
##      [,1] [,2]       [,3]
## [1,] -1.0    0  0.6666667
## [2,]  0.5    0 -0.1666667
## [3,] -0.5    1 -0.5000000
```

We can also get some other important information from this matrix, like the sums for each column and each row.

```
colSums(mymatrix)
```

```
## [1]  3  7 11
```

```
rowSums(mymatrix)
```

```
## [1]  9 12
```

Matrices

Matrices can work with our other objects:

```
x <- 2
mymatrix+x
```

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

```
mymatrix*x
```

```
##      [,1] [,2] [,3]
## [1,]    2    6   10
## [2,]    4    8   12
```

Matrices

Recall that R does not default to vector or matrix operations, so to get R to do matrix multiplication, we use `x%*%y` instead of `x*y`.

```
mymatrix %*% mymatrix2
```

```
##      [,1] [,2] [,3]
## [1,]   22   49    3
## [2,]   28   64    4
```

Note: `x*y`, with `x` and `y` vectors or matrices, will either return an error or R will perform a pointwise multiplication.

Data

We can do a lot with the few things we've already learned! But now we want to think about applying those to data.

To set up a working directory where all of your packages and other R files will be downloaded to / uploaded from

```
getwd()
```

```
## [1] "C:/Users/GradQuant/Dropbox (UCR GradSuccess)/UCR GradSuccess Team Folder/2018-19 UCR GradSuccess/GradQua
```

```
setwd("C:\\Users\\GradQuant\\Desktop")
getwd()
```

```
## [1] "C:/Users/GradQuant/Desktop"
```

Data

We can import different types of data

```
data = read.table("hmnrghts.txt", header=TRUE)
data = read.csv ("hmnrghts.csv", header=TRUE)
```

In RStudio, we can do this using "Import Dataset" in the "Environment" tab.

fig:

R also has a huge number of built in datasets (most of which are in specific packages). We can use these to test functions or to practice our R skills.

```
data(mtcars)
```

If we want to find out about the mtcars dataset

```
?mtcars
```

Data

We'll rename this "data" so that it's easier to work with

```
data <- mtcars
```

To view the first few rows

```
head(data)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

or the last few rows

```
tail(data)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

The `head` and `tail` functions are a nice way to make sure that your data was imported correctly without trying to print out too much. This is especially important for larger datasets.

Data

Say we wanted to view just one column. We might be temped to try

```
mpg
```

but that won't work.

Instead, we need to be specific about where our column comes from

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

That's more like it! The `$` command tells R that we want to extract the named element `mpg` out of `mtcars`.

Data

Alternately, we can "attach" our dataset. This will make our dataset R's top priority. Now, we can refer to items in our dataset directly.

```
attach(data)
mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

Notice that you can only attach one dataset at a time!

If we don't need a certain dataset anymore, we can detatch it

```
detach(data)
```

Data

We can also select columns by number. Say we want the first three. We can use either format:

```
newdata <- data[,c(1,2,3)]
newdata <- data[,c(1:3)]
```

```
head(newdata)
```

```
##                    mpg cyl disp
## Mazda RX4         21.0   6  160
## Mazda RX4 Wag     21.0   6  160
## Datsun 710        22.8   4  108
## Hornet 4 Drive    21.4   6  258
## Hornet Sportabout 18.7   8  360
## Valiant           18.1   6  225
```

Data

If we want columns that are not adjacent to each other, say the first and third through sixth,

```
newdata<-data[,c(1,3:6)]
head(newdata)
```

```
##                    mpg disp  hp drat    wt
## Mazda RX4         21.0  160 110 3.90 2.620
## Mazda RX4 Wag     21.0  160 110 3.90 2.875
## Datsun 710        22.8  108  93 3.85 2.320
## Hornet 4 Drive    21.4  258 110 3.08 3.215
## Hornet Sportabout 18.7  360 175 3.15 3.440
## Valiant           18.1  225 105 2.76 3.460
```

Data

We can also use this concept to remove variables by using a minus sign:

```
droppeddata <- data[,c(-3,-5)]
head(droppeddata)
```

```
##                    mpg cyl  hp    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 110 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 110 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  93 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 110 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 175 3.440 17.02  0  0    3    2
## Valiant           18.1   6 105 3.460 20.22  1  0    3    1
```

Or a more concise way for multiple columns (now we remove three through five, instead of three and five)

```
newdata <- data[,c(-3:-5)]
```

We can select certain rows the same way

```
newdata <- data[1:5,]
```

Data

Another important means of data manipulation is subsetting. Say we want all cars with miles per gallon greater than 30.

```
subset(data, mpg>30)
```

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

The subset function takes in the original vector, matrix, or data that you want to subset as the first argument. The second argument is the conditions for subsetting.

Exercise

Select all the cars with an mpg greater than 20.0 and engine displacement over 200, and name it `exercisedata`. Then output the `mpg`, `disp`, and `am` columns.

Exercise Solution

```
exercisedata <- subset(data,mpg > 20)
exercisedata <- subset(exercisedata, exercisedata$disp > 200)
exercisedata[,c("mpg","disp","am")]
```

```
##                 mpg disp am
## Hornet 4 Drive 21.4  258  0
```

Notice that when we subset `exercisedata`, we needed to use that `$` again. This is because `data` is attached, not `exercisedata`.

We can also do this faster by having R check both conditions in one line of code:

```
exercisedata <- subset(data,mpg > 20 & disp>200)
exercisedata[,c("mpg","disp","am")]
```

```
##                 mpg disp am
## Hornet 4 Drive 21.4  258  0
```

Using R Packages

Base R has a lot of functionality, but the real power comes from it's open-sourced nature. User-written packages greatly expand R's capabilities, but they need to be installed and loaded.

```
install.packages("matrixStats")
library(matrixStats)
```

With a package loaded, we can use our help function to examine its documentation

```
?matrixStats
```

All packages should have documentation, but some have much better documentation than others!

Statistics in R

Now let's do some basic statistics on our mtcars dataset…this is what R is built for!

Let's start by finding the mean and standard deviation

```
mean(data$mpg)
```

```
## [1] 20.09062
```

```
sd(data$mpg)
```

```
## [1] 6.026948
```

Statistics in R

If we want more complete information, we can use

```
summary(data$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   10.40   15.43   19.20   20.09   22.80   33.90
```

or something like

```
psych::describe(data$mpg)
```

```
##    vars  n  mean   sd median trimmed  mad  min  max range skew kurtosis
## X1    1 32 20.09 6.03   19.2    19.7 5.41 10.4 33.9  23.5 0.61    -0.37
##      se
## X1 1.07
```

Notice how we were able to use a function from a package without loading it - we called the `psych` package as part of the `describe` command.

Statistics in R: A Simple t-Test

The V/S variable tells us whether the car is a V-engine or a straight engine. We will compare mpg based on the V/S variable.

First, we need to make sure that `vs` is a factor (categorical variable)

```
is.factor(data$vs)
```

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

It's not, so we need to convert it

```
data$vs <- as.factor(data$vs)
```

Let's try it again

```
is.factor(data$vs)
```

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

That's better.

Statistics in R: A Simple t-Test

Now, using that `describe` function again, we can examine the data broken down by group

```
psych::describeBy(mpg, group=vs)
```

```
##
##  Descriptive statistics by group
## group: 0
##    vars  n  mean   sd median trimmed  mad  min max range skew kurtosis
## X1    1 18 16.62 3.86  15.65   16.42 2.97 10.4  26  15.6 0.48    -0.05
##      se
## X1 0.91
## --------------------------------------------------------
## group: 1
##    vars  n  mean   sd median trimmed mad  min  max range skew kurtosis
## X1    1 14 24.56 5.38   22.8   24.34   6 17.8 33.9  16.1 0.41     -1.4
##      se
## X1 1.44
```

Statistics in R: A Simple t-Test

Finally, we can do our t-test

```
t.test(mpg~vs, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  mpg by vs
## t = -4.6671, df = 22.716, p-value = 0.0001098
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.462508  -4.418445
## sample estimates:
## mean in group 0 mean in group 1
##        16.61667        24.55714
```

Way significant!

Statistics in R: Linear Models

We can so the same analysis in the form of a linear model (here we're showing a regression model). This gives the same results as our t-test. (Phew!)

```
model <- lm(mpg~vs, data)
summary(model)
```

```
##
## Call:
## lm(formula = mpg ~ vs, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.757 -3.082 -1.267  2.828  9.383
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   16.617      1.080  15.390 8.85e-16 ***
## vs1            7.940      1.632   4.864 3.42e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.581 on 30 degrees of freedom
## Multiple R-squared:  0.4409, Adjusted R-squared:  0.4223
## F-statistic: 23.66 on 1 and 30 DF,  p-value: 3.416e-05
```

Statistics in R: ANOVA

We can run an ANOVA for more than two groups. We have three cylinder groups. They aren't saved as factors, but we have a shortcut:

```
anova <- aov(mpg~as.factor(cyl), data)
summary(anova)
```

```
##                Df Sum Sq Mean Sq F value   Pr(>F)    
## as.factor(cyl)  2  824.8   412.4    39.7 4.98e-09 ***
## Residuals      29  301.3    10.4                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Statistics in R: ANOVA

We can also extract an ANOVA from a model:

```
model <- lm(mpg~as.factor(cyl), data)
anova <- aov(model)
summary(anova)
```

```
##                Df Sum Sq Mean Sq F value   Pr(>F)    
## as.factor(cyl)  2  824.8   412.4    39.7 4.98e-09 ***
## Residuals      29  301.3    10.4                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

These two ANOVA tables are exactly the same! So why might we want the model?

Statistics in R: Linear Models

Let's take a look at all the elements in our model

```
names(model)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"         
##  [5] "fitted.values" "assign"        "qr"            "df.residual"  
##  [9] "contrasts"     "xlevels"       "call"          "terms"        
## [13] "model"
```

This tells us the different elements that we can get from R's storage of our model.

Statistics in R: Linear Models

For instance, we can check the residuals…

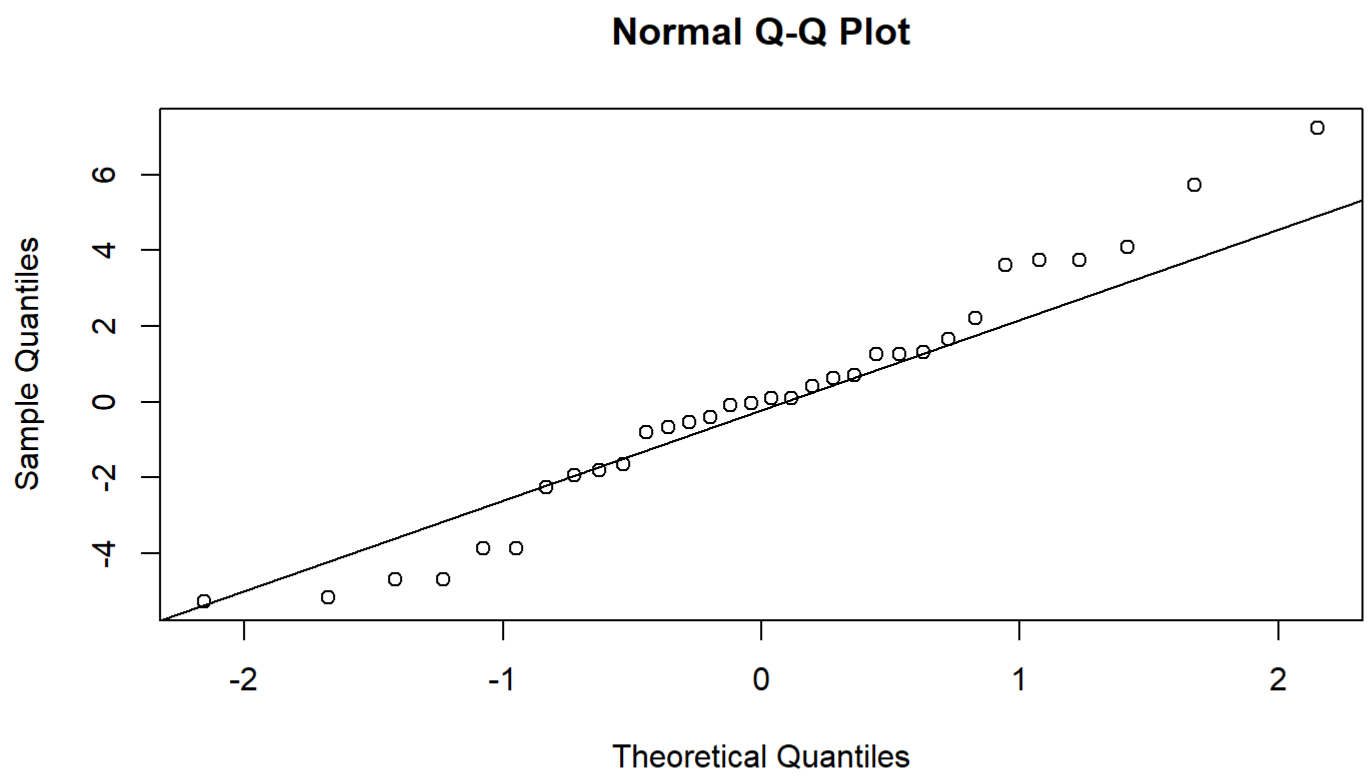
```
model$residuals
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##          1.25714286          1.25714286         -3.86363636
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##          1.65714286          3.60000000         -1.64285714
##          Duster 360           Merc 240D            Merc 230
##         -0.80000000         -2.26363636         -3.86363636
##            Merc 280           Merc 280C          Merc 450SE
##         -0.54285714         -1.94285714          1.30000000
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##          2.20000000          0.10000000         -4.70000000
## Lincoln Continental   Chrysler Imperial            Fiat 128
##         -4.70000000         -0.40000000          5.73636364
##         Honda Civic      Toyota Corolla       Toyota Corona
##          3.73636364          7.23636364         -5.16363636
##    Dodge Challenger         AMC Javelin          Camaro Z28
##          0.40000000          0.10000000         -1.80000000
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##          4.10000000          0.63636364         -0.66363636
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##          3.73636364          0.70000000         -0.04285714
##       Maserati Bora          Volvo 142E
##         -0.10000000         -5.26363636
```

Statistics in R: Linear Models

But these aren't very interesting on their own. Let's check the residuals for normality using a qqplot.

```
qqnorm(model$residuals)
qqline(model$residuals)
```



Statistics in R: Correlations

Say we wanted to correlate mpg with engine displacement.

```
cor(mpg, disp)
```

```
## [1] -0.8475514
```

but that doesn't give much output. Let's try this instead

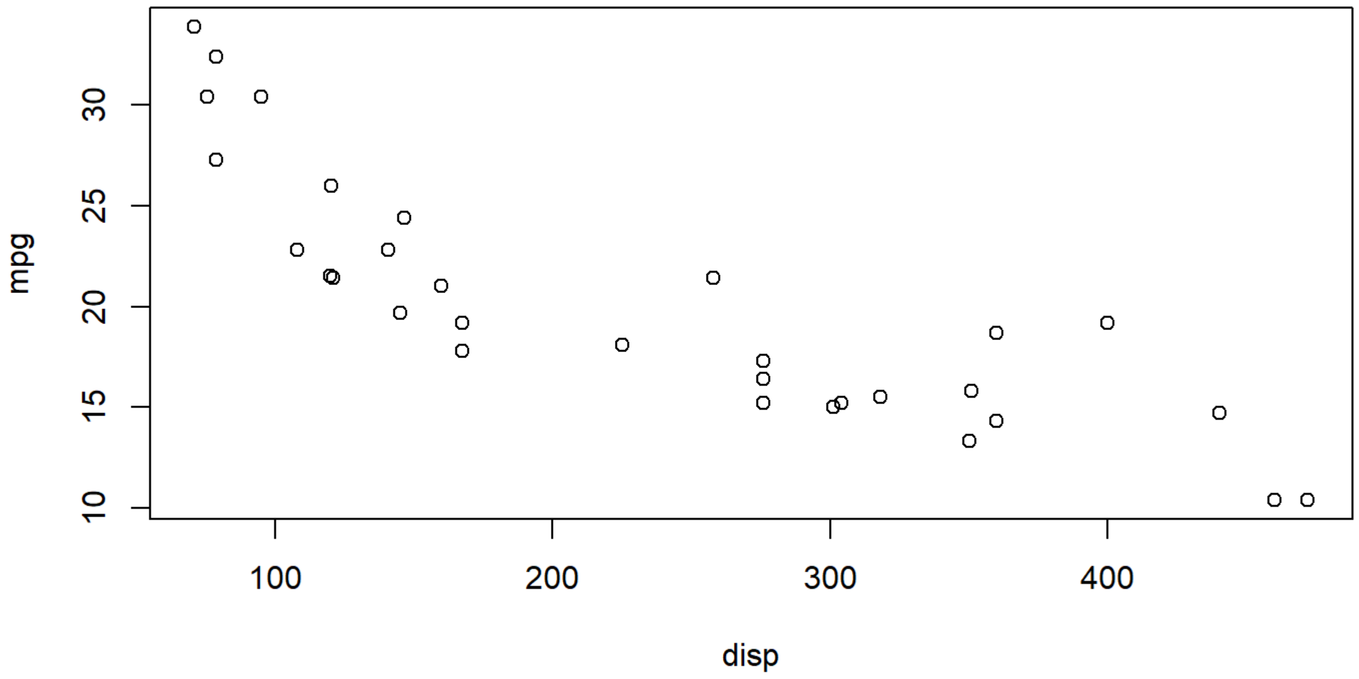
```
cor.test(mpg, disp)
```

```
## 
##  Pearson's product-moment correlation
## 
## data:  mpg and disp
## t = -8.7472, df = 30, p-value = 9.38e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9233594 -0.7081376
## sample estimates:
##        cor 
## -0.8475514
```

Plots

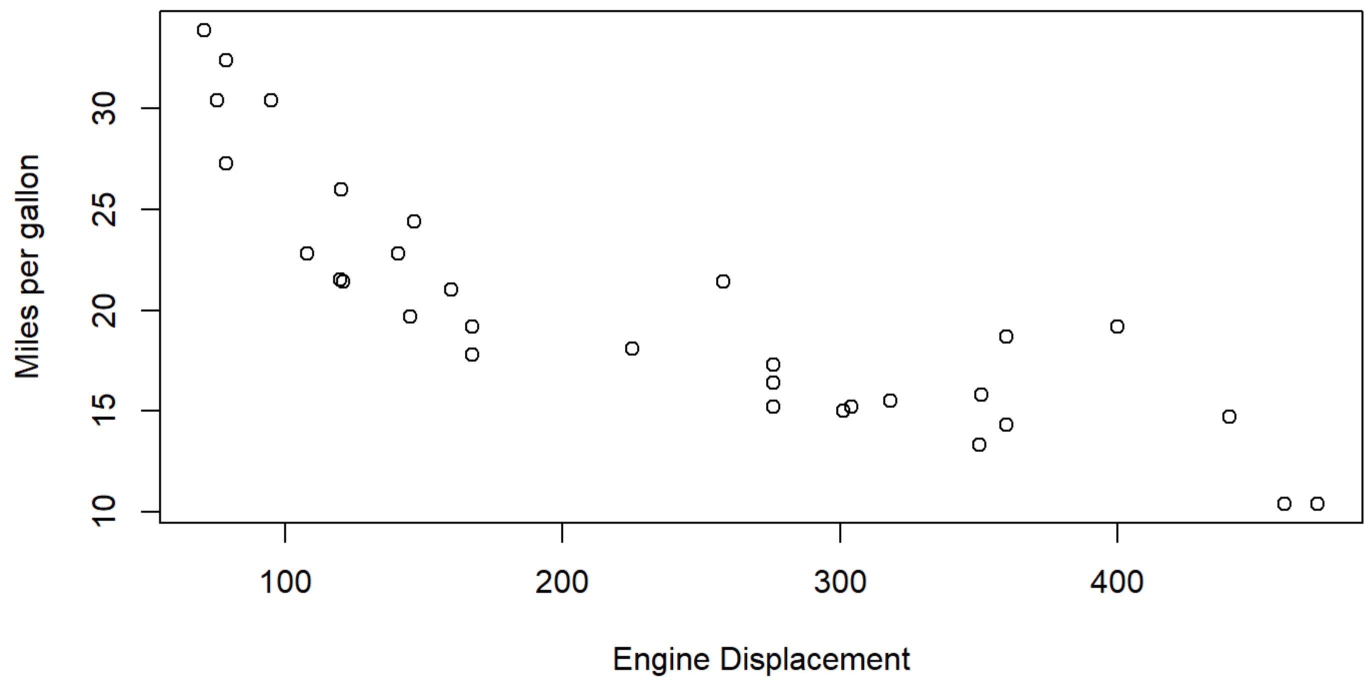
We got a little glimpse of R's plotting with `qqplot`, but R can do a lot more than that. Let's start simple:

```
plot(mpg~disp)
```



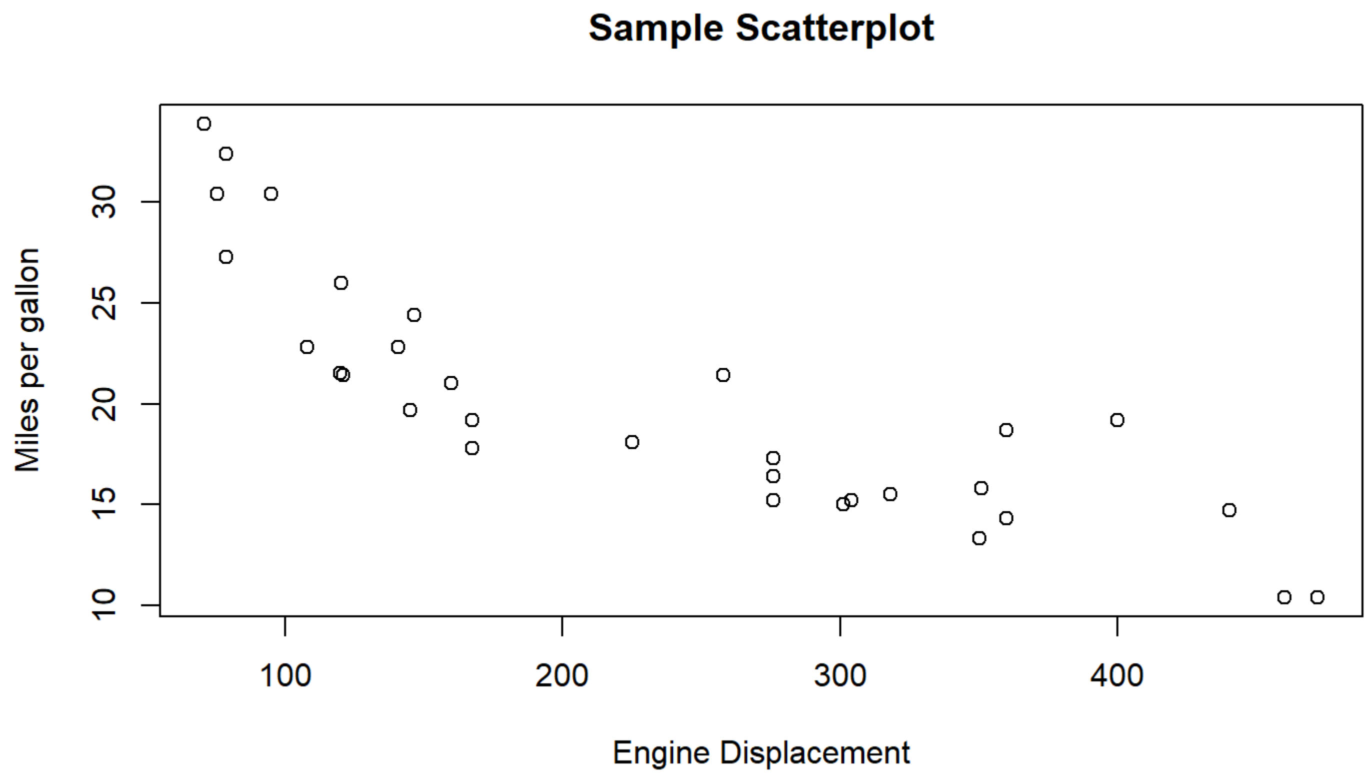
It's a nice start, but we can do so much more. Let's start with better labels

```
plot(mpg~disp, xlab = "Engine Displacement", ylab="Miles per gallon")
```



…and a main title

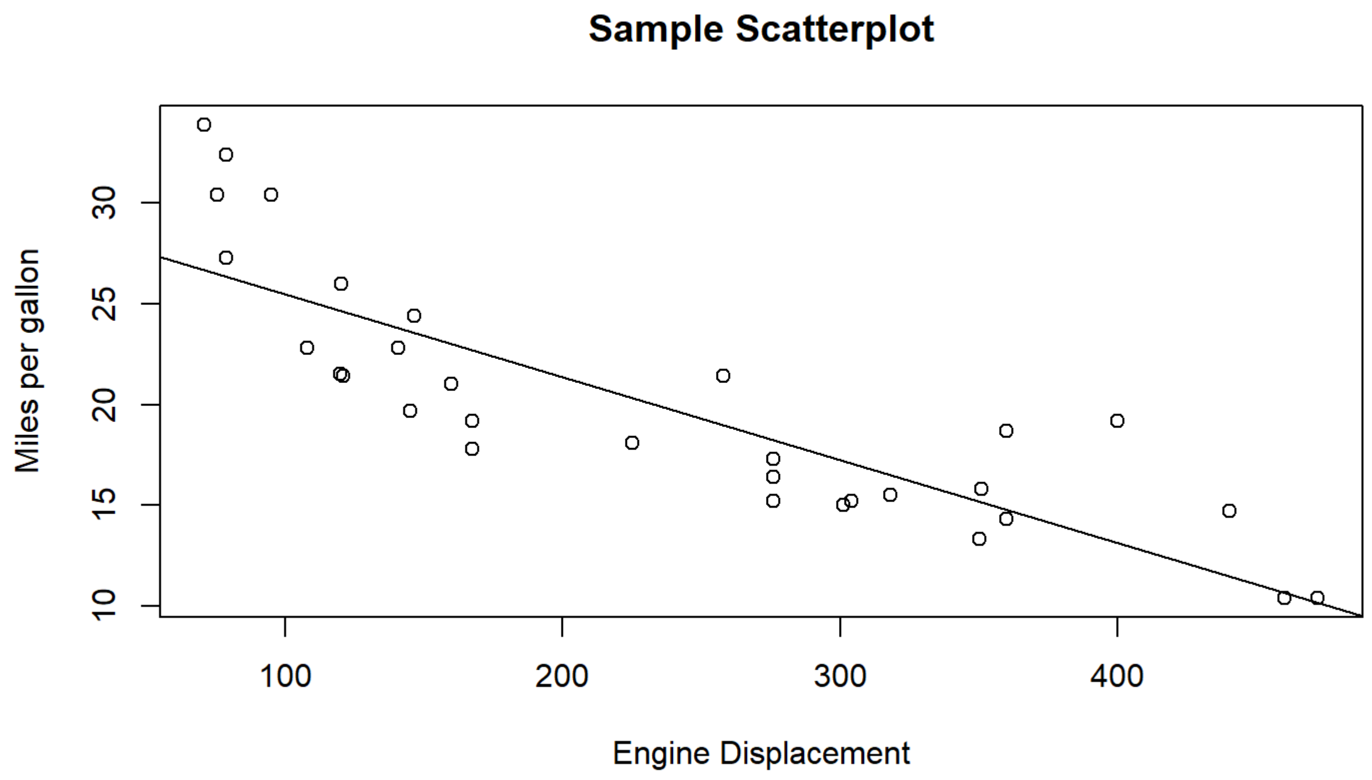
```
plot(mpg~disp, xlab = "Engine Displacement", ylab="Miles per gallon",
     main="Sample Scatterplot")
```



51/59

…and maybe even a best fit regression line

```
plot(mpg~disp, xlab = "Engine Displacement", ylab="Miles per gallon",
     main="Sample Scatterplot")
abline(lm(mpg~disp))
```



Exercise

Find the correlation between weight (`wt`) and quarter mile time (`qsec`).

Then run a linear model, and create a scatterplot.

Exercise: Solutions

```
cor(wt, qsec)
```

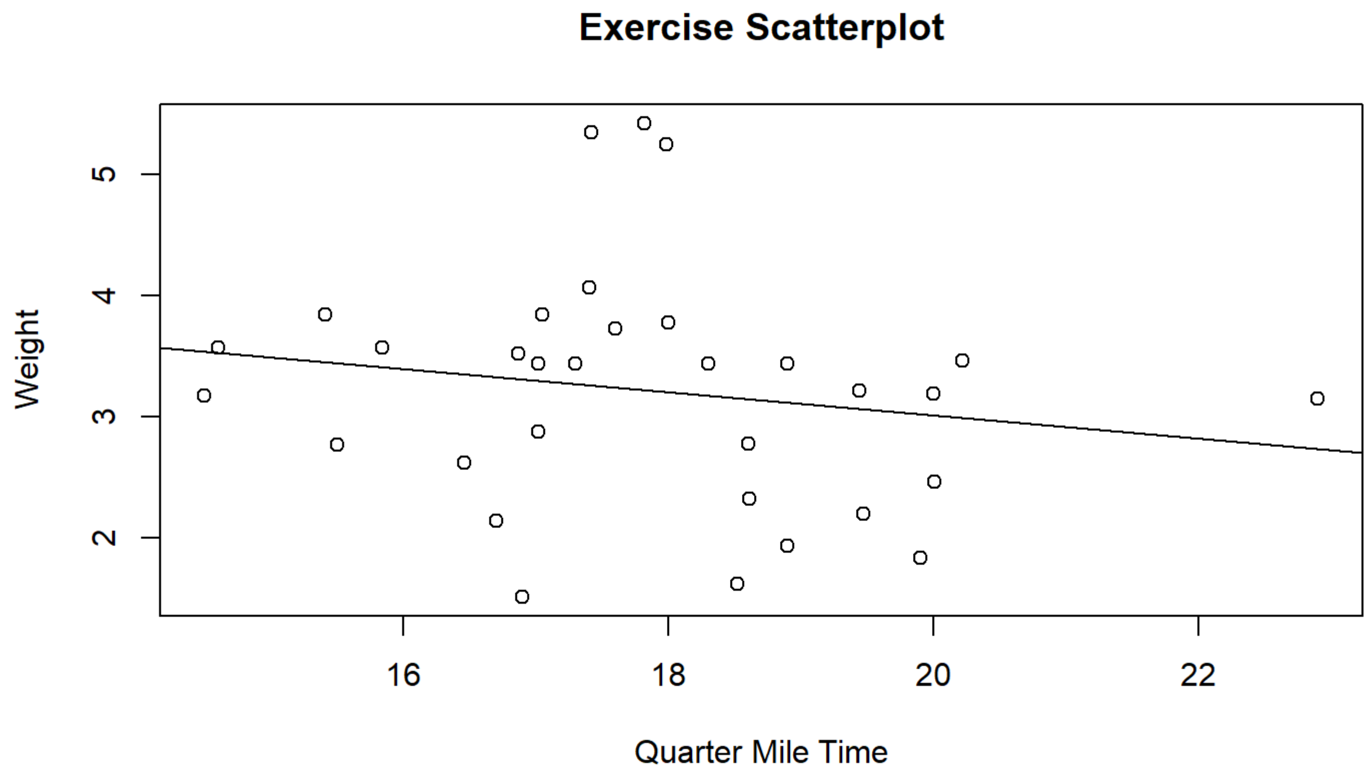
```
## [1] -0.1747159
```

```
lm(wt~qsec)
```

```
##
## Call:
## lm(formula = wt ~ qsec)
##
## Coefficients:
## (Intercept)         qsec
##     4.92479     -0.09567
```

Exercise: Solutions

```
plot(wt~qsec, xlab = "Quarter Mile Time", ylab="Weight",
     main="Exercise Scatterplot")
abline(lm(wt~qsec))
```



Statistics in R: Categorical Variables

Finally, let's examine the relationship between two categorical variables. First, we'll create a contingency table

```
table <- xtabs(~vs + am)
table
```

```
##    am
## vs   0  1
##   0 12  6
##   1  7  7
```

and do a chi-squared test on it

```
chisq.test(table)
```

```
## 
##  Pearson's Chi-squared test with Yates' continuity correction
## 
## data:  table
## X-squared = 0.34754, df = 1, p-value = 0.5555
```

It's as easy as that!

Functions

Now that we've used them some, let's dissect the anatomy of a function…

```
?seq
```

All of the options associated with a particular function are called "arguments". Some arguments have defaults that we can override. Others don't have a default and we may need to specify some input.

If we have the right order, we don't need to specify the argument

```
lm(mpg~vs,data)
```

```
##
## Call:
## lm(formula = mpg ~ vs, data = data)
##
## Coefficients:
## (Intercept)          vs1
##       16.62         7.94
```

Functions

If we get the order wrong, it returns an error (or does the wrong thing!)

```
lm(data, mpg~vs)
```

But we can use any order we like if we specify the argument

```
lm(data=data, formula = mpg~vs)
```

```
##
## Call:
## lm(formula = mpg ~ vs, data = data)
##
## Coefficients:
## (Intercept)          vs1
##       16.62         7.94
```

Thanks For Coming!

There's more to learn about R, but that's all we have time for today.

We will be having more intermediate R workshops this school year! Previous years' workshops have included "Graphing in R" and "Data Manipulation in R". We plan to do similar workshops soon.

If you have questions in the meantime, please feel free to schedule a consultation or come to drop-in hours!