```
# BUDT 758T: Spring 2019
# Instructor: Courtney Paulson

## In the Review of R Coding file, we used the Auto data set from the ISLR
library
library(ISLR)
attach(Auto)
View(Auto)

### Loops: for and while

## "Looping" in coding refers to going through a set of data and applying
## the same actions repeatedly to different data points
## For example, let's say I want to create a new variable called Eight_Cyl
## Eight_Cyl is "YES" if a car has 8 cylinders and "NO" if a car does not
## ifelse() will do this for us:

Eight_Cyl=ifelse(cylinders==8,"YES","NO")
head(Eight_Cyl)
unique(Eight_Cyl)

## I could do this in a loop, though, by checking every row of data
## First, I need to know how many rows of data there are

last=nrow(Auto)
last

## Create a new variable that is the same length as our data
## The rep() function repeats the first argument a given number of times
## So this will repeat the value "0" 392 times:

Eight_Cyl2=rep(0,last)

## Now, set up a "for loop"; we want it to run through every row from 1 to the
last row
## And for each row, record if the car has eight cylinders or not

for(i in 1:last){

  if(cylinders[i]==8){Eight_Cyl2[i]="YES"}
  else{Eight_Cyl2[i]="NO"}

}

head(Eight_Cyl2)

## A "while loop" keep running as long as a certain condition is met
## For example, let's say we want to know where the first 6-cylinder car is
## This while loop will keep running as long as cylinders[i] does not equal 6
## We cycle through each row of data by increasing i by 1 each time
## But make sure you restart i, because otherwise i will start at 393!
```

```
i=1
while(cylinders[i]!=6){

  i=i+1
}

i
cylinders[i]

## Exercise on your own: We could also have done the same thing we did
## with ifelse() and for() with the while loop.
## Write a while loop for the Eight_Cyl3 variable below so Eight_Cyl3[i]="YES"
## if the car has 8 cylinders and "NO" if not

Eight_Cyl3=rep(0,last)



# Extra Info: Loading Data

### These commands might be helpful if you have to use base R (rather than
RStudio)
### Keep in mind, though, that this will try to read files from whatever your
working directory is!
## The Beer dataset is a very common one for introducing data mining techniques
## We'll see it in practice next week!

beer=read.csv("Beer.csv",header=T)
dim(beer)
names(beer)

## Handy option in R: data editor!

fix(beer)
```

```
## What if your data set had NA values (missing data)?

beer2=read.csv("Beer_missing.csv",header=T)
dim(beer2)
names(beer2)

## Let's say we just want to check if every entry is/is not missing

is.na(beer2)

## Probably impractical for large data sets.
## Instead, let's check how many missing values do we have?

sum(is.na(beer2))
mean(is.na(beer2))
sapply(beer2, function(x) sum(is.na(x)))

## But is all our data actually good data? Let's run summary stats

summary(beer2)

## We have some problematic values! Gender should only be 0 and 1
## And it is illegal for anyone under the age of 21 (in the US) to drink beer
## So let's update our data:

beer2$Age[beer2$Age < 21] <- NA
beer2$Gender[(beer2$Gender != 0 & beer2$Gender != 1)] <- NA
beer2$Married[beer2$Married != 0 & beer2$Married != 1] <- NA

sapply(beer2, function(x) sum(is.na(x)))

## Easiest way to deal with missing data: delete it

beer_nomiss=na.omit(beer2)
dim(beer_nomiss)

## Frequently, however, we don't want to just eliminate missing data
## To impute missing data (replace missing with non-missing value)
## Consider the mice library. Basic tutorials can be found here:
## https://datascienceplus.com/imputing-missing-data-with-r-mice-package/
## and https://datascienceplus.com/handling-missing-data-with-mice-package-a-
simple-approach/

## Note that we usually use categorical variables like "Preference"
## as dummy variables (0 and 1). We could do this with another ifelse()
statement:

Pref_dummy=ifelse(beer$Preference=="Regular",1,0)
Pref_dummy

## Or we could change the type:
```

```r
Pref_dummy = as.numeric(beer$Preference)
Pref_dummy

## What happened here? R always starts from 1 (and uses alphabetical order)
## To fix it:

Pref_dummy = as.numeric(beer$Preference)-1
Pref_dummy

## Exercise on your own: Load the Exams_Missing.csv file and answer the following:
## 1) How many total missing values are there?
## 2) How many missing values are in each variable?
## 3) Each exam score should be between 0 and 100. Change any other scores to missing.
##      How do your answers to 1 and 2 change?
##      Helpful hint: The | symbol means "OR" (like & means "AND")
```

```r
# Example: Simple Linear Regression

## The Review file contained some fairly boring analyses
## As you saw in Data Models, we can do more exciting analyses as well
## You've already seen linear regression in previous classes, so check out how R does it

## First, use the glm() function to run the linear model and save the results
## You may have learned lm() before instead of glm()
## glm() is for generalized linear models--we can use it for all regressions, not just linear!
## But the default is still linear regression
## We get other regression options by changing parameters in glm()
## Regardless, you always enter your regression as glm(Y~X)

lm.fit=glm(horsepower~mpg)
summary(lm.fit)
```

```
## names() will show you everything stored in the lm.fit data structure

names(lm.fit)
lm.fit$coefficients
coef(lm.fit)

## You can use this structure to easily plot a best fit line using abline()
## An abline() is always added TO a plot, so first do a plot

plot(mpg,horsepower,xlab="MPG",ylab="HP")
abline(lm.fit)
abline(lm.fit,lwd=3)
abline(lm.fit,lwd=3,col="red")

plot(mpg,horsepower,col="red")
plot(mpg,horsepower,pch=20)
plot(mpg,horsepower,pch="+")

## If you want to show more than one plot at a time, you can change the
parameters of the plot function
## The glm() function automatically saves four basic residual/model assumption
check plots

par(mfrow=c(2,2))
plot(lm.fit)

## Exercise on your own: Using the beer data set, try to predict a person's
## Income using their Age and the glm() function. Does there appear to be a
relationship? Why or why not?
## Report the plot of the two variables, with the regression line in green









# Example: Running Multiple Linear Regression

## You run an MLR very similarly to an SLR. Just add more X variables by
literally adding them to the function:

lm.fit=glm(horsepower~mpg+weight)
summary(lm.fit)
```

```
## Doing a "." instead of putting in specific variables will give you all
remaining variables:

lm.fit=glm(horsepower~.)

## If you do this, though, make sure you put in a data set!

lm.fit=glm(horsepower~.,data=Auto)
summary(lm.fit)

## Again, using a negative sign will remove variables from the
command/regression:

lm.fit=glm(horsepower~.-name, data=Auto)
summary(lm.fit)

## You can also do variable transformations and calls inside lm() rather than
changing the data itself

summary(glm(horsepower~.-name+I(weight^2),data=Auto))
summary(glm(horsepower~.-name+log(weight),data=Auto))

## Two options for interactions:

summary(glm(horsepower~mpg*weight,data=Auto))
summary(glm(horsepower~.-name+mpg:weight,data=Auto))
```