# Regression Summary Output

```r
require(mosaic)
```

In this lab we will learn how to compute and interpret all of the numbers that show up in the standard regression output and ANOVA table. Moreover, we will learn how to perform a simple linear regression without actually using the `lm` command.

Alongside those tasks, we're going to be ramping up our `dplyr` skills, using the `mutate()` and `summarize()` commands, and building larger chains.

For this lab, we will examine data from the Environmental Protection Agency on the fuel economy. Note that since this data set comes from the `fueleconomy` package, we can load it with `data` command.

```r
# install.packages("fueleconomy")
require(fueleconomy)
data(vehicles)
```

It's a good idea to read the documentation for this data set before working it, so that you know what the variables are actually measuring. In particular, this *should* tell you important things like: "in what units are the variables measured?" and "how are the categorical variables encoded?"

```r
help(vehicles)
```

We'll restrict our attention to cars in the 2000 model year with 4 cylinder engines.

```r
myCars <- vehicles %>%
  filter(year == 2000 & cyl == 4)
```

We want to model the fuel economy (measured in miles per gallon) as a function of the size of the engine (measured in cubic centimeters). We should always begin by investigating a scatterplot of the two variables.

```r
# Note the use of transparency (the "alpha" option) in order to show multiple points that are right on
xyplot(hwy ~ displ, data=myCars,
       main="Fuel Economy", alpha=0.5, cex=2, pch=19,
       xlab="Engine Size (cubic centimeters)",
       ylab="Fuel Economy (miles per gallon)")
```

The pattern seems roughly linear, with a couple of exceptions, so a linear model seems like a reasonable approach.

### The actual regression output

We first construct our linear model, and examine the standard regression output produced by R.

```r
m1 <- lm(hwy ~ displ, data=myCars)
summary(m1)
```

Our goal for the rest of this lab is to pretend that we don't know `m1`, and to reproduce each of the quantities listed in the output.

### Finding beta1 using SXX and SXY

Since we are performing a simple linear regression, we can find the equation for the regression line in just a few steps. Of course, which set of steps you choose to take will depend on how you code in `R`! We'll go through a few equivalent ways.

One approach starts by computing the differences from the means for both our response and explanatory variables,

```
regdata <- myCars %>%
  mutate(xdif = displ - mean(displ),
         ydif = hwy - mean(hwy))
```

The `mutate()` command lets us add new columns to our data. Once you've run this, look at the data (either by clicking on it in RStudio, using `View()`, or using `head()`) to see what the new variables look like.

Then, we can compute our SXX (sum of squares of the explanatory variable) and our SXY (co-variation for the explanatory and response variable together).

```
regdata <- regdata %>%
  summarize(SXX = sum(xdif^2),
            SXY = sum(xdif*ydif))
```

This action needs the `summarize()` command, because it is taking a large data set and summarizing it down to something smaller. `mutate()`, on the other hand, always gives you the same number of rows, just additional columns.

The coefficient $\hat{\beta}_1$ is just the ratio of SXX to SXY.

```
regdata <- regdata %>%
  mutate(beta1=SXY/SXX)
regdata
```

Lets check to make sure our coefficient is the same as the one from the internal R function.

```
coef(m1)["displ"]
```

Note that this computation was pulled apart for into many steps for you. But, `dplyr` lets you chain together many data operations, so I could also have written

```
myCars %>%
  mutate(xdif = displ - mean(displ),
         ydif = hwy - mean(hwy)) %>%
  summarize(SXX = sum(xdif^2),
            SXY = sum(xdif*ydif),
            beta1=SXY/SXX)
```

Or, I could have used different functions altogether,

```
myCars %>%
  summarize(n=n(),
            SXX = var(displ) * (n-1),
            SXY = cov(hwy,displ) * (n-1),
            beta1 = SXY/SXX)
```

### Finding beta1 using correlation coefficient

An even easier way to compute the slope of the regression line is to use the correlation coefficient between the response and explanatory variable. The slope of the regression line is equal to the correlation coefficient, scaled by the ratios of the standard deviations of the response and explanatory variables

$$\hat{\beta}_1 = r_{XY} \cdot \frac{s_Y}{s_X}$$

```
myCars %>%
  summarize(beta1 = cor(hwy, displ) * (sd(hwy) / sd(displ)))
```

### Finding beta0

Now that we know the slope of the regression line, we can compute the intercept using the point-slope formula for a line that you learned in high school. We can do this because the point $(\bar{x}, \bar{y})$ **always** lies on the regression line.

```
regdata <- myCars %>%
  summarize(beta1 = cor(hwy, displ) * (sd(hwy) / sd(displ)),
            meanX = mean(displ),
            meanY = mean(hwy))

# Estimate the intercept, using the fact that the means
# define a point on the regression line
regdata %>%
  mutate(beta0 = meanY - beta1 * meanX)
```

Thus, our regression equation is:

$$\hat{hwy} = \hat{\beta}_0 + \hat{\beta}_1 \cdot displ$$

As promised, the means lie on the regression line.

```
predict(m1, newdata=data.frame(displ=mean(~displ, data=myCars)))
mean(~hwy, data=myCars)
```

### Assessing the Fit

Now that we have the regression model, we want to assess how well it fits the data. The first step in that process is to partition the variance in the response variable into two parts: that which is explained by the model, and that which is not.

```
# We're going to need differences from the mean down the line, so lets start by computing them
assessdata <- myCars %>%
  mutate(ydif = (hwy - mean(hwy)))
```

In order to compute the portion of the variation explained by the model, we need to generate the fitted values.

```
assessdata <- assessdata %>%
  mutate(fitted = fitted(m1))
```

Now we can compute the sums of squares

```
assessdata <- assessdata %>%
  summarize(n = n(),
            SST = sum(ydif^2),
            SSE = sum((fitted - hwy)^2),
            SSM = sum((fitted - mean(hwy))^2))
```

and check that they add up

```
assessdata %>%
  mutate(SSE + SSM)
```

Again, we could have chained all this computation together, like so:

```
myCars %>%
  mutate(ydif = (hwy - mean(hwy)),
         fitted = fitted(m1))  %>%
  summarize(SST = sum(ydif^2),
            SSE = sum((fitted - hwy)^2),
            SSM = sum((fitted - mean(hwy))^2))
```

With all the sums of squares in hand, we are now ready to compute the **coefficient of determination** ($R^2$):

```
# Coefficient of determination
assessdata <- assessdata %>%
  mutate(rsq = 1 - SSE / SST)
```

Is it the same?

```
rsquared(m1)
```

The adjusted $R^2$ imposes a penalty for additional explanatory variables. This is necessary since adding superfluous variables will usually cause $R^2$ to go up, and we want a model that is both simple, and effective.

```
# p is the number of explanatory variables
p <- 1

assessdata <- assessdata %>%
  mutate(adjrsq = 1 - (SSE / (n-1-p)) / (SST / (n-1)))
```

## Hypothesis tests for significance of explanatory variables

Once again, we're going to need a bunch of values from the data, so I'll start by pulling together some pieces from earlier in the lab.

```
testdata <- myCars %>%
   mutate(ydif = (hwy - mean(hwy)),
         fitted = fitted(m1)) %>%
  summarize(n=n(),
            meanX = mean(displ),
            meanY = mean(hwy),
            SXX = var(displ) * (n-1),
            SXY = cov(hwy,displ) * (n-1),
            beta1 = SXY/SXX,
            beta0 = meanY - beta1 * meanX,
            SST = sum(ydif^2),
            SSE = sum((fitted - hwy)^2),
            SSM = sum((fitted - mean(hwy))^2))
```

Before we compute the values in the table for the explanatory variables, we need to know the **residual standard error**. Now that we've pulled together basically every number we have computed so far, this is easy.

```
# Residual Standard error
testdata <- testdata %>%
  mutate(RSE = sqrt(SSE / (n-2)))
```

Now we can perform hypothesis tests on the slope of the regression line. Recall that the null hypothesis in this case is that $\beta_1 = 0$. The **standard error** for the slope coefficient is

```r
# Standard error
testdata <- testdata %>%
  mutate(SE1 = RSE / sqrt(SXX))
testdata %>% glimpse()
```

which immediately leads to the $t$-statistic

```r
# t-statistic
testdata <- testdata %>%
  mutate(t1 = beta1 / SE1)
testdata %>% glimpse()
```

and the $p$-value

```r
# p-value
testdata %>%
  summarize(p = 2 * pt(abs(t1), df=(n-2), lower.tail = FALSE))
```

The computation for the intercept term is quite similar

```r
# Compute statistics for the intercept
# Standard error
testdata <- testdata %>%
  mutate(SE0 = RSE * sqrt((1/n) + (meanX)^2 / SXX))
# t-statistic
testdata <- testdata %>%
  mutate(t0 = beta0 / SE0)
testdata %>% glimpse()
# p-value
testdata %>%
  summarise(p = 2 * pt(abs(t0), df=(n-2), lower.tail = FALSE))
```

### ANOVA table

The last remaining computation is the $F$-statistic. It is easier to see the components of this by looking at the ANOVA table.

```r
anova(m1)
```

We compute the $F$-statistic by taking the ratio of the mean sum of squares. Since we have only one explanatory variable in this case, we have

```r
# F-statistic
testdata <- testdata %>%
  mutate(F = (SSM / p) / (SSE / (n-1 - p)))
```

The $p$-value for the $F$-statistic is equal to the area in the right-hand tail of the $F$-distribution with `p` and `n-p-1` degrees of freedom.

```r
testdata %>%
  summarize(p = pf(F, df1 = p, df2 = n-1 - p, lower.tail=FALSE))
```

### Nested F-tests

We often make a series of models with the same response variable, and want to know the significance of the various subsets of explanatory variables. To do this, we use a nested F-test.

For this exercise, we will consider some data on blood pressure. Researchers observed the following data on 20 individuals with high blood pressure:

- blood pressure (`BP`, in mm Hg)
- age (`Age`, in years)
- weight (`Weight`, in kg)
- body surface area (`BSA`, in `m^2`)
- duration of hypertension (`Dur`, in years)
- basal pulse (`Pulse`, in beats per minute)
- stress index (`Stress`)

Our goal is to build a model for blood pressure as a function of (some subset) of the other variables. In this case all of our variables are quantitative, so we can get a quick look at their relationships using the a pairs plot.

```r
bloodp <- read.csv("https://andyreagan.github.io/teaching/2018/09-SDS-291/data/bloodpress.csv")
# install.packages("GGally")
library(GGally)
ggpairs(bloodp)
# pairs(bloodp) # this is a little faster, but uglier
```

Without any intution, one way to proceed is to simply throw all of the variables into our regression model. In a sense, we are throwing everything but the kitchen sink into the model. Using the `.` in the formula interface includes all non-response variables in the data frame

```r
mfull <- lm(BP ~ ., data=bloodp)
summary(mfull)
```

We have a very high $R^2$, and as such the correlation between the fitted values and the response variable is very strong. However, there are still some issues with the normality of the residuals. Moreover, the coefficient for `Pulse` is negative, suggesting that people with higher pulse rates have lower blood pressure. [Does this make sense?] Three of the coefficients in the model are close to zero and not significant at the 10% level.

One problem here could be multicolinearity, which (recall) we can study using VIF.

```r
require(car)
vif(mfull)
```

So, we might want to make some other models. Let's study the following models:

```r
m1 <- lm(BP ~ Weight, data=bloodp)
m2 <- lm(BP ~ Weight +  Age, data=bloodp)
m3 <- lm(BP ~ Weight +  Age + Dur + Stress, data=bloodp)
```

Notice that these models are all subsets of `mfull`, so we can determine which model is best using a nested F-test.

```r
# Add the models in ascending order of complexity.
anova(m1, m2, m3, mfull)
```

- Given this output, which model would you choose?

Like the regression summary above, we want to make sure we understand where the values come from in the output. Let's try computing some of the numbers in the table ourselves.

Let's consider an ANOVA table for just two of the regression models.

```r
anova(m2, mfull)
```

```r
SSM_full = sum((fitted.values(mfull) - mean(~BP, data=bloodp))^2)
SSM_reduce = sum((fitted.values(m2) - mean(~BP, data=bloodp))^2)
```

```
SSM_full - SSM_reduce

SSE_full = sum(residuals(mfull)^2)
SSE_full

((SSM_full - SSM_reduce)/4)/(SSE_full/(20-6-1))
```

Now, the question is whether we can do this on the larger scale.