Lab 3: The Linear Model in R

Eric Parajon

Adapted from code and text from Isabel Laterzo, Simon Hoellerbauer and Chelsea Estancona.

This lab will introduce you to `lm()`, the function we will be using to fit linear models (for now) in `R`. I will walk you through the different parts of this lab, but then we will largely be working independently. Chime in whenever you have questions!

Don't forget to set your working directory!

```
setwd()
```

Part 1: LM in R

First, we'll simulate some data to use. This will be a very useful skill in the future, so pay attention to what we do here.

We first create our variables and then stipulate what our effects should be.

Let's set our seed to 123.

For our $\mathbf{x}_1$ variable, let's take 20 draws from a $\mathcal{N}(1.3, 2.3)$ (using the `R` parameterization, where the second parameter is the standard deviation).

Let's set β_0 to be 17, and β_1 to be 1.3. We also can't forget to simulate $\boldsymbol{\epsilon}$! Let's $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 3.4)$.

```
set.seed(123)

##Explanatory variables
x1 <- rnorm(__, __, __)

##Coefficients
b0 <- 17 #Intercept
b1 <- 1.3 #Beta 1

##standard deviation
sigma <- 3.4

error <- rnorm(length(x1), 0, sigma) #Let's simulate some random error, too.
```

What would we do in order to simulate a linear relationship with the variables and effects above? Create the outcome variable `y`.

```
y <- ____________________
```

What do $\mathbf{x}_1$ and $\mathbf{y}$ look like together? Use the `plot` function to plot a scatter plot of $\mathbf{x}_1$ against $\mathbf{y}$.

```
plot(x1, y)
```

Now that we have simulated explanatory variables and set up a 'true' relationship between our explanatory variable (x1) and outcome variable, we can use R's built-in functions to evaluate our linear model. For a

linear model, we will want to use the `lm()` function. The basic syntax for this function is `lm(Y ~ X1 + X2)`.

Use `summary()` on the model object (`m1` in this case) in order to see some of the most interesting information about the fitted model.

```
m1 <- lm(___________) #the basic syntax here: lm(Y ~ X1 + X2)
#summarize the 'model object' we've created
summary(m1)
```

Part 2: Combining LM and the tidyverse

We can also combine what we learned about using the tidyverse with linear modeling!

As we dive into learning linear models in `R`, we're going to use some data from the `mtcars`. But first, lets use the `tidyverse` to clean it up a bit. First, take a look at the mtcars data (its readily available so you don't really need to load it in!).

```
library(tidyverse)
#examine data
summary(mtcars)
```

Great. Let's say we want to analyze the role that `disp` and `cyl` play in influencing `mpg`. However, we're only interested in observations where `cyl` is greater than 4. Let's use `dplyr` and the `tidyverse` to only keep observations where `cyl` is greater than 4. Then, only keep the three above variables in the new data set. Label this data set `new_data`.

Hint: the `dplyr` functions you are interested in here are `filter()` and `select()` - don't forget to call their help files if you need more information (e.g., `?filter()` in your console).

```
new_data <- mtcars %>%
  filter(cyl > 4) %>%
  select(__, __, __)
```

Now that we have our data ready, we can use `lm()`. Fill in the blank in the chunk below where Y is `mpg`, X1 is `cyl`, and X2 is `disp`. Use `summary()` on the model object (`m2` in this case) in order to see some of the most interesting information about the fitted model.

Remember, our data is `new_data`.

```
#model
m2 <- lm(___________)

#summarize
summary(___)
```

The `summary` function provides lots of valuable information about the model we've created. We can also extract different information if we need, for example, just the coefficients, or just the residuals. Use `View()` on `m2`. What does it look like `m2` is? What do you think we could do if we wanted to access the different objects within `m2`?

```
#Taking a look at m1
------------

#extract coefficients
m2$_____

#extract residuals
m2$______
```

```
#extract y hat
m2$_______
```

We can also use indexing to get to parts of these vectors - the output from a model is a list object, which means we can use brackets with $ indexing.

```
m2$coefficients[1] #What does this give us?
m2[2] #What about this?
```

What if we wanted to include an interaction term? Try interacting `disp` and `cyl`.

```
m3 <- lm(mp ~ disp + cyl + _______, data=new_data) #use * for interaction- include both variables

summary(m3)
```

We don't actually have to write all of the variables two times; the `lm` function will include the variables involved in the interaction even if we don't specify them individually in the formula. Try this out below, creating model object `m4` and compare it to `m3`.

```
#m4:
m4<-lm(_________)
```

How might we present these results in a table? Both `xtable()` and `stargazer()` are functions that produce LaTeX code that could be copy-pasted into a .tex document or included in an Rmarkdown document. Try both of these on one of the model object `m4` we created above and see what happens. Note, these are VERY flexible and can be customized to make very nice tables, the ones we are generating here are just basic.

```
#These produce code for a .pdf that we can copy and paste
xtable(______)
stargazer(_______)

#you can also add lots of useful information onto these
#check out guides online!
```

When presenting our results, we might be largely concerned with prediction: if I have a model and provide it with 'new' data, what should I expect? How confident are we in that expectation? We can use the handy `predict()` function here, which works with almost any type of model object.

Check out `predict.lm` using ?, as that is the method that `predict` invokes when we feed it an lm object.

```
#fitting our model
m5 <- lm(mpg ~ disp + cyl, data = mtcars)

#looking at our model
summary(m5)

#what are we doing here? Also, what does with allow us to do here?
x_vec_n <- with(mtcars, seq(min(disp), max(disp), length.out = 500))
#we've created a 500 length vector starting from the min value of disp
#through the max value of disp

#why am I choosing these values? What are other options?
new_dat_n_1 <- with(mtcars, data.frame(x_vec_n, 4))
new_dat_n_2 <- with(mtcars, data.frame(x_vec_n, 6))
new_dat_n_3 <- with(mtcars, data.frame(x_vec_n, 8))

#what are we doing here?
```

```r
colnames(new_dat_n_1) <- c("disp", "cyl")
colnames(new_dat_n_2) <- c("disp", "cyl")
colnames(new_dat_n_3) <- c("disp", "cyl")
#we just created three data frames - all with our x_vec_n vaalues for "disp"
#and three values we have set for "cyl"

#this is very important for predict! See note below.


#getting our predicted values
fit_n_1 <- predict(m5, newdata = new_dat_n_1, interval = "confidence")

fit_n_2 <- predict(m5, newdata=new_dat_n_2, interval="confidence")

fit_n_3 <- predict(m5, newdata=new_dat_n_3, interval="confidence")

#ggplot is rather particular about the format of the data you will be using
plot_dat_n <- data.frame(fit_n_1, fit_n_2, fit_n_3, new_dat_n_1)

#note we never call `cyl` on its own below so the iteration of `new_dat_n_x` we include
#here doesn't matter a ton, it just gives us all values of disp

#this looks like a monster plot, but there are just a lot of layers. A lot of
#ggplot plot code will end up looking like this
ggplot(data=plot_dat_n) +
  labs(x = "Displacement of Vehicle", y = "Miles Per Gallon")+
  geom_line(aes(x = disp, y = fit_n_1[, 1], color = "4 cylinders"),linetype = "solid") +
  geom_line(aes(x = disp, y = fit_n_2[, 1], color = "6 cylinders"),linetype = "solid") +
  geom_line(aes(x = disp, y = fit_n_3[, 1], color = "8 cylinders"), linetype = "solid") +
  #what are we doing with the ribbons? What does alpha do? Play around with it.
  geom_ribbon(aes(x = disp, ymin = fit_n_1[, 2], ymax = fit_n_1[, 3]), alpha = .3, fill = "red") +
  geom_ribbon(aes(x = disp, ymin = fit_n_2[, 2], ymax = fit_n_2[, 3]), alpha = .3, fill = "green") +
  geom_ribbon(aes(x = disp, ymin = fit_n_3[, 2], ymax = fit_n_3[, 3]), alpha = .3, fill = "blue") +
  theme(legend.title = element_text(size = 14),
        legend.position = "bottom",
        panel.grid.minor.x = element_blank(), #what does this do?
        panel.grid.minor.y = element_blank()) +
  scale_color_discrete(name="Number of Cylinders") + #what does this seem to do? what happens if you ta
  theme_bw()
```

Note that when using predict, it is very important that our new observations have variables that bear the same name as in the formula. Otherwise `predict` won't know what to do! Try out the following:

```r
#recreating new_dat_n_1
new_dat_n_1_a <- with(mtcars, data.frame(x_vec_n, 4))

#but this time we don't label the variables
predict(m5, newdata = new_dat_n_1_a, interval = "confidence")

#and this time we label them incorrectly
colnames(new_dat_n_1_a) <- c("cisp", "dyl")
predict(m5, newdata = new_dat_n_1_a, interval = "confidence")

#and this time we flip the variables
```

```
new_dat_n_1_a <- with(mtcars, data.frame(4, x_vec_n))
colnames(new_dat_n_1_a) <- c("disp", "cyl")
predict(m5, newdata = new_dat_n_1_a, interval = "confidence")
#this time it works, but our predictions are clearly incorrect (compare them to
#fit_n_1 from above)

#this works!
new_dat_n_1_a <- with(mtcars, data.frame(4, x_vec_n))
colnames(new_dat_n_1_a) <- c("cyl", "disp")
fit_n_1_a <- predict(m5, newdata = new_dat_n_1_a, interval = "confidence")

all(fit_n_1 == fit_n_1_a) #what does all() do here?
```

Part 3: LM practice

Using the palmerpenguines dataset, create an OLS using multiple predictors and then display the results in a publication quality table (using stargazer).

```
library(palmerpenguins)
library(stargazer)
```