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!

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 $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 3.4)$.

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

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.

Part 2: Combining LM and the tidyverse

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).

Now that we have our data ready, 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)`. 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 (`m1` in this case) in order to see some of the most interesting information about the fitted model.

Remember, our data is `new_data`.

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`?

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.

What if we wanted to include an interaction term?

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`.

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.

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.

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:

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).