Economics Club Introduction to R Example

Jason Smith - GVSU Economics Club

Introduction

This is the companion document for the Introduction to R code that was presented at the Economics Club meeting. My intention for this is to be some form of "director's cut" where I can explain a bit in a bit more detail what's going on. Some alterations have been made to the code to better show things in document form. I think the best way to learn R is by using R. I hope this document is helpful jumping off point.

Getting Help

R is unlike any software package most people have used in the past. While R give us the freedom to perform more complex analysis than programs like Excel, it's also easier for new users to make mistakes. When your code won't run, try not to get frustrated. Every error is an opportunity to learn something new about R and will ultimately make you a better programmer. You can't get better if you don't make mistakes!
When you get an error, read the error message carefully. They can often help you determine what the problem is. You can also copy/paste the message into Google and find help online. You're probably not the first person to run into that error!
This may not sound like useful advice, but it's so important to at the very least skim through the documentation for a function if you don't know how to use it. In R you can do this in the console by typing `?FUNCTION_NAME` and pressing enter. This will pull up the documentation in the bottom right. The documentation will tell you what arguments the function accepts and how they work. There are often examples at the very bottom of the page. You can also find documentation at www.rdocumentation.org.
The internet also has a lot of resources for learning R. Googling what you're trying to do in R or the package you're working with will often provide instructions and code examples. Not all of it will be relevant for what you're doing in the moment so it's important to read for a variety of sources and experiment to find what works.

Data Background

The data used in this document comes from the *palmer penguin* package available www.github.com/allisonhorst/palmerpenguins. It contains body measurements for 344 penguins. The codebook follows

Table 1: Codebook

Variable	Meaning
Species	The species of the penguin (Adelie, Chinstrap, Gentoo)
island	Island where penguin lives (Biscoe, Dream, Torgenson)
bill.length.mm	Length of the bill in millimeters
bill.depth.mm	Depth (or height) of the bill in millimeters
body.mass.g	Body mass of the penguin in grams
sex	Sex of the penguin (male, female)
year	Year the measurement was taken (2007, 2008, 2009)

Loading Packages

Packages are bits of shareable code. Typically, they'll contain datasets or new functions so you don't have to program them yourself. The most common way to install a package is with the command `install.packages("PACKAGE_NAME")`. You can then load the package to be used in your code with the command `library(PACKAGE_NAME)`. In the majority of cases, you want to start your code by loading all the packages you'll be using. By doing this, you ensure your packages are properly loaded before you have to use them. It also makes it easy for a collaborator to see what packages you're using.

This code uses the following packages.

- tidyverse: a collection of packages designed for data science
- texreg: summarizes regression output in publication ready tables
- vtable: easily creates publication ready tables
- ggpubr: wrapper for the *ggplot2* packages (included in the tidyverse) for publication ready graphics

```
library(tidyverse)
library(texreg)
library(vtable)
library(ggpubr)
```

Reading in Data

R can read data in a variety of formats. The easiest way to load your data is using the *import data* button in the environment tab. The wizard will walk you through browsing for the file and importing it into the environment. You can then copy and paste the import code into the script file.

```
penguin <- read_csv("data/penguin.csv")
```

Data Summaries

When you get a new dataset, a good way to begin to understand it is with summary statistics. Before we start with that, we probably first want to know how intact the data is. That is, do we have missing data? In this dataset, we can click on the *penguin* entry in the environment tab to open it in a new window. In this case, we can quickly see some values are missing (noted in R by *NA*). To show this in document form, below is some code that will create a table with the sum of all the *NA* entries in the dataset. We find that there are 17 *NA*s. Please note that this code isn't counting how many observations (penguins in the case) have missing data, just how many values are missing in total. So it would be improper to say that we have missing data for 19 penguins.

```
penguin %>%
  summarise(sum_na = sum(is.na(penguin))) %>%
  knitr::kable(caption = "Missing Values") %>% #extra code for PDF output
  kable_styling(latex_options = "hold_position")
```

Table 2: Missing Values

sum_na
19

There are a lot of ways to handle missing data but the easiest is probably to remove them. To do this, we'll use the `drop_na()` function from the *tidyr* package (included in the tidyverse). Notice how we have to reassign the data to the environment after I drop the *NA*s. In this case, we're just overwriting the original penguin dataset in the environment with the version without missing values.

```
penguin <- drop_na(penguin)
```

We can confirm the missing values are gone by asking R to sum up the *NA* values like we did before.

```
penguin %>%
  summarise(sum_na = sum(is.na(penguin))) %>%
  knitr::kable(caption = "No NAs") %>% #Extra code for PDF output
  kable_styling(latex_options = "hold_position")
```

Table 3: No NAs

sum_na
0

To make our summary statistics tables look a bit nicer, I want to separate the categorical and quantitative variables. We'll assign two new variables (categorical and quant) to the environment. Using the `select()` function from the *dplyr* package (include in the tidyverse), we'll select which columns we want to include in each new variable.
Notice how the `select()` function first asks us for the data we want to select from (penguins), then for the variables within the dataset we want to include.
For the sake of convenience, we're not including the variable *year* into either variable. Dates can be difficult to work with. If you would like to learn how to work with dates and times, I recommend reading chapter 16 in *R for Data Science* which covers the *lubridate* package.

```
categorical <- select(penguin, species, island, sex)
quant <- select(penguin, bill.length.mm, bill.depth.mm, body.mass.g)
```

Now that our data is separated, we'll create summary tables using the `sumtable()` function from the *vtable* package. These tables will appear in the *viewer* pane in the bottom right be default. From there, you can export the table as an image by clicking the *export* button at the top of the viewer pane.

```
sumtable(categorical, title = "Summary Statistics for Categorical Variables") %>%
  kable_styling(latex_options = c("striped", "hold_position")) #Extra code for PDF output
```

Table 4: Summary Statistics for Categorical Variables

Variable	N	Percent
species	333	
... Adelie	146	43.8%
... Chinstrap	68	20.4%
... Gentoo	119	35.7%
island	333	
... Biscoe	163	48.9%
... Dream	123	36.9%
... Torgersen	47	14.1%
sex	333	
... female	165	49.5%
... male	168	50.5%

If we wanted to export the table as a .csv file to further edit in Excel, we could provide additional arguments. Set *out = "csv"* and *file = "file/path/to/save/location.csv"*. For example, we could save the table to a folder called *output* in the working directory with the command `sumtable(penguin, out = "csv", file = "output/summary_stats.csv)`.

```
sumtable(quant, title = "Summary Statistics for Quantitative Variables") %>%
  kable_styling(latex_options = c("striped", "hold_position")) #Extra code for PDF output
```

Table 5: Summary Statistics for Quantitative Variables

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
bill.length.mm	333	43.993	5.469	32.1	39.5	48.6	59.6
bill.depth.mm	333	17.165	1.969	13.1	15.6	18.7	21.5
body.mass.g	333	4207.057	805.216	2700	3550	4775	6300

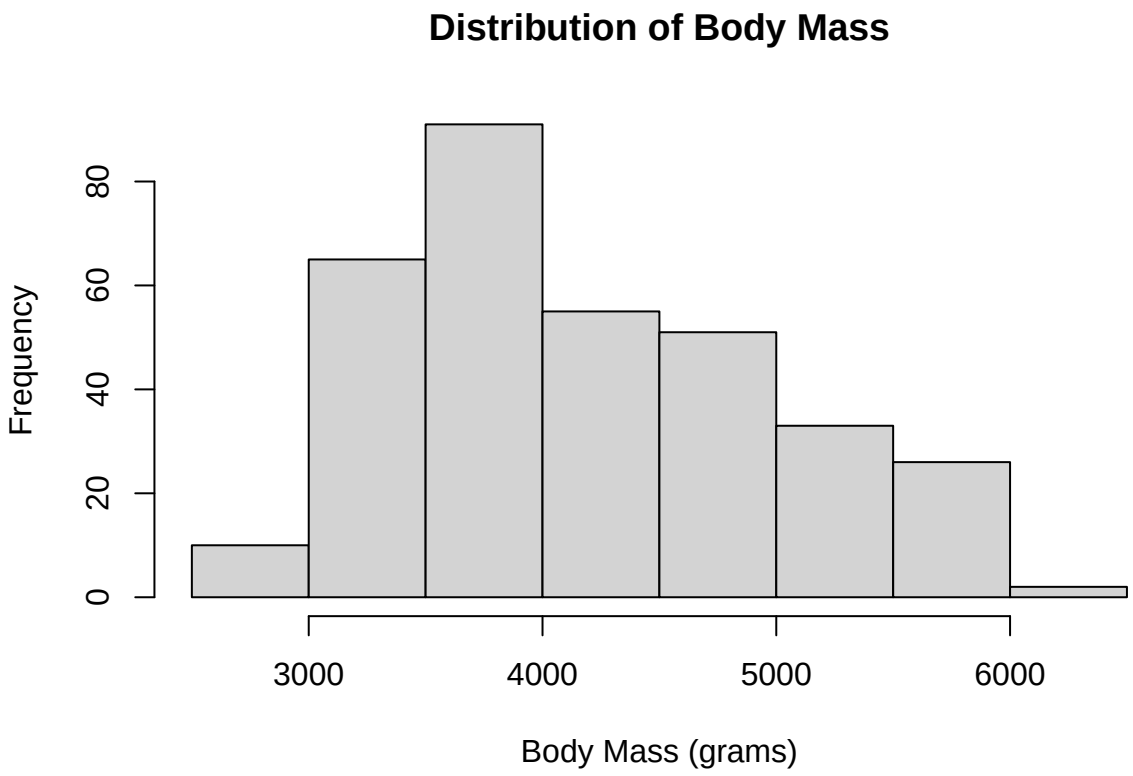
Plotting

Graphs are another way to gain insight from data. In some cases, graphs can help us see subtleties that we would not have otherwise noticed in other formats like a table. As they say, a picture's worth a thousand words!

Let's plot the distribution of penguin body mass with a histogram. In base R (no packages needed), we can do this with the `hist()` function. This function asks us specifically which variable in our dataset we want to plot. We can specifiy this with the `$`. We'll extract the *body.mass.g* variable from the penguin dataset by saying `penguin$body.mass.g`. We'll also change the name of the title and x-axis with the *main* and *xlab* arguments respectively.

The resulting graph will appear in the *plots* pane in the bottom right. we can save it by clicking the *export* button.

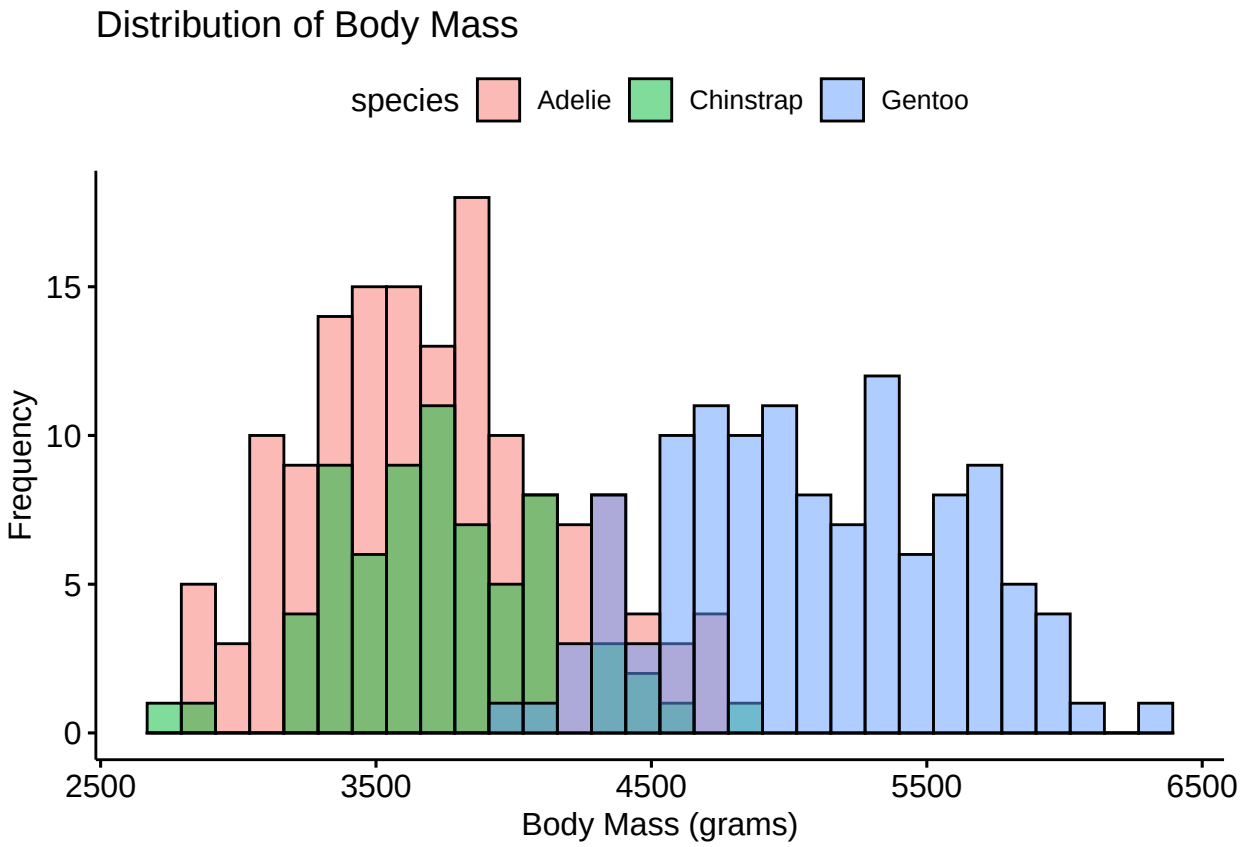
```
hist(penguin$body.mass.g, main = "Distribution of Body Mass", xlab = "Body Mass (grams)")
```



The base plotting looks fine, but we can make more complex and professional-looking figures using the *ggplot2* package (included in the tidyverse). ggplot2 can be a bit daunting at first, so we're going to be using the *ggpubr* package which is a wrapper for ggplot2. We'll give up some customization options in exchange for a more straight-forward experience. If you'd like to learn how to use the full ggplot2 package (and I recommend you do), chapter 3 of *R for Data Science* is a great starting point.

To make a histogram, we'll call the `gghistogram()` function. This function asks us for our data and the variable seperately so we don't have to use the `$`. We'll add a bit of complexity to this graph by setting `fill = "species"`. This will group the data by species and create a histogram for each one. The resulting figure will be all the histograms stacked on top of each other. This is a good way to see how the distribution of body weight varies based on species. Like before, the graph will appear in the *plots* pane where it can be exported.

```
gghistogram(data = penguin, x = "body.mass.g", fill = "species",
            title = "Distribution of Body Mass",
            ylab = "Frequency",
            xlab = "Body Mass (grams)")
```



Regression

Fitting a regression model is also a common way to understand data. It allows us to begin to quantify how variables interact and explain each other. We'll fit a linear model using the `lm()` function. this function first asks us for the formula of the model we want to fit in the form `y ~ x1 + x2 + ... +xi`, then the data where the variables are located. We assign this model to a variable in the environment so we can use it later.

```
lm <- lm(body.mass.g ~ flipper.length.mm + sex, data = penguin)
```

The most rudimentary way to see the results of the regression is to use the `summary()` function. This function will give you the usual regression output, but it can be difficult to read and is not publication ready.

```
summary(lm)
```

```
##
## Call:
## lm(formula = body.mass.g ~ flipper.length.mm + sex, data = penguin)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -910.28 -243.89   -2.94  238.85 1067.73
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -5410.300    285.798 -18.931  < 2e-16 ***
## flipper.length.mm    46.982      1.441  32.598  < 2e-16 ***
## sexmale             347.850     40.342   8.623 2.78e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 355.9 on 330 degrees of freedom
## Multiple R-squared:  0.8058, Adjusted R-squared:  0.8047
## F-statistic: 684.8 on 2 and 330 DF,  p-value: < 2.2e-16
```

This is where a package like *texreg* comes in. This package contains a number of functions that will produce easier to read regression output in a variety of formats.
To start, we have the `screenreg()` function. This just prints the output to the console. It's not publication ready but is much easier to read than `summary()`

```
screenreg(lm)
```

```
##
## ================================
##                     Model 1
## --------------------------------
## (Intercept)        -5410.30 ***
##                     (285.80)
## flipper.length.mm     46.98 ***
##                      (1.44)
## sexmale              347.85 ***
##                      (40.34)
## --------------------------------
## R^2                    0.81
## Adj. R^2               0.80
## Num. obs.            333
## ================================
## *** p < 0.001; ** p < 0.01; * p < 0.05
```

To get a publication ready output, we can use the `texreg()` or `htmlreg()` function. `texreg()` will generate the LaTeX code for the table. `htmlreg()` will generate the html code for the table. I'm using `texreg()` for this document since the document is written in LaTeX.

```
texreg(lm)
```

	Model 1
(Intercept)	-5410.30^{***}
	(285.80)
flipper.length.mm	46.98^{***}
	(1.44)
sexmale	347.85^{***}
	(40.34)
R^2	0.81
Adj. R^2	0.80
Num. obs.	333

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

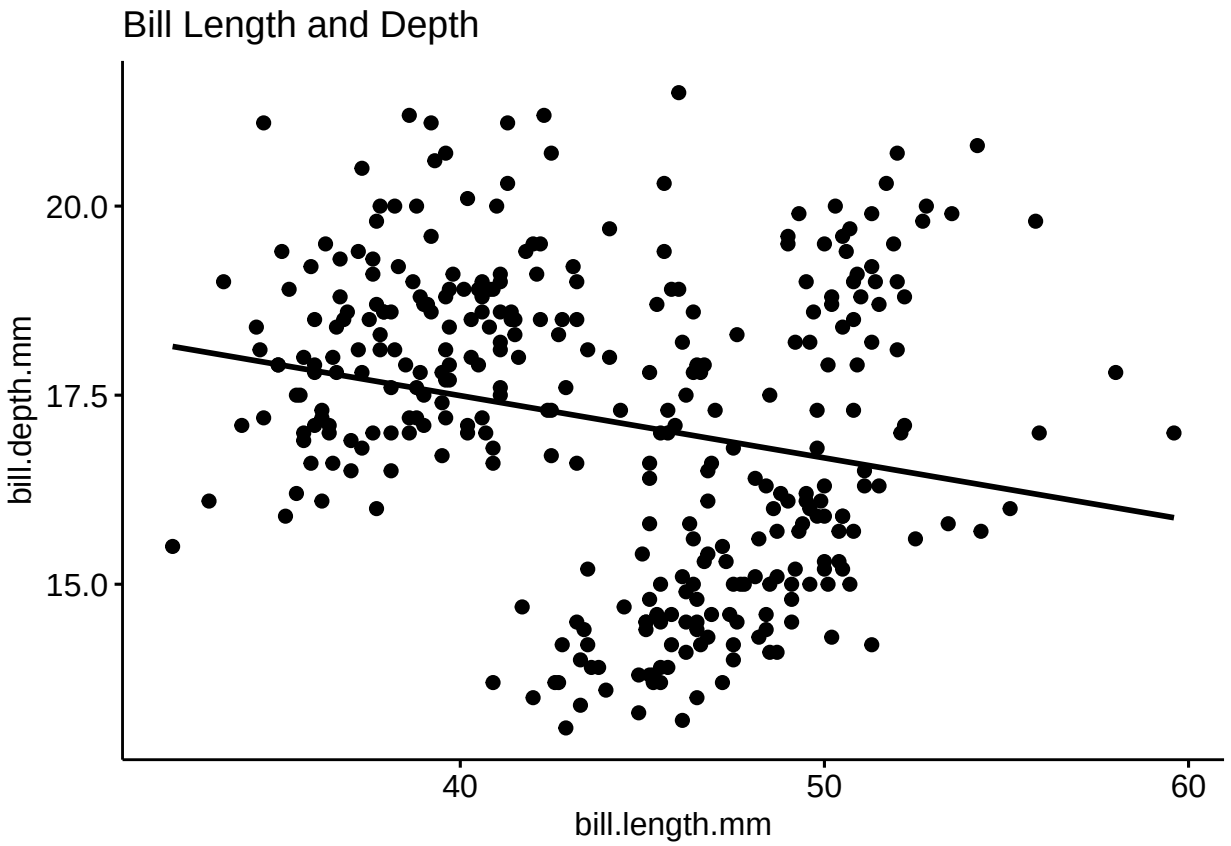
Table 6: Statistical models

The far easier approach will be to use `htmlreg()` and save the document as a .doc file that can be opened in Word.
The code would look something like `htmlreg(model, file = "file/path/name.doc")`

Truthfulness and Helpfulness

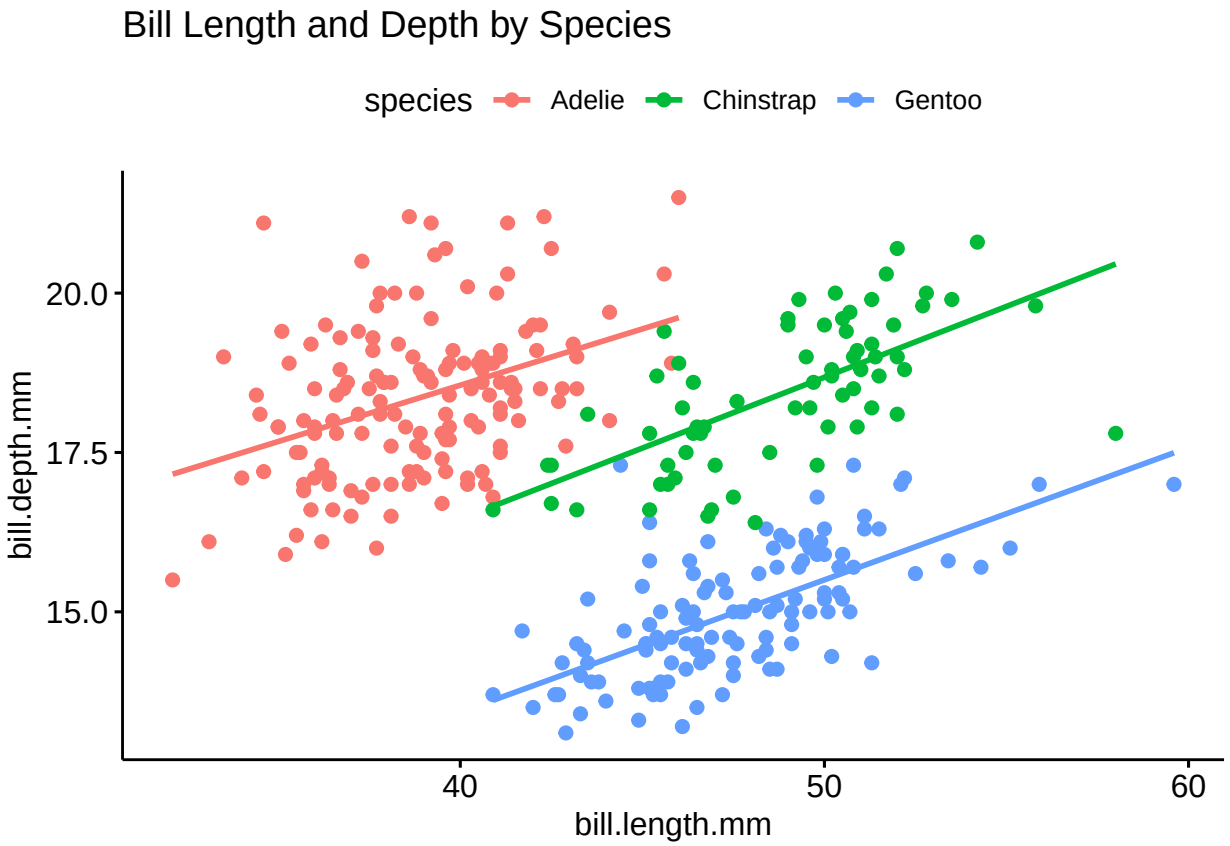
The final thing this document will cover is a reminder that you fully explore the data before making conclusions.
Say we want to examine the relationship between the length and depth of a penguin's bill. If we're not interested in the regression coefficients, we can add the argument `add = "reg.line"` to `ggscatter()` to fit the regression line `y ~ x` to the scatter plot (if we wanted the regression coefficients, we'd have to use the `lm()` function discussed above).

```
ggscatter(data = penguin, x = "bill.length.mm", y = "bill.depth.mm",
          add = "reg.line",
          title = "Bill Length and Depth")
```



The regression shows a negative relationship between bill length and depth. That's a bit odd. Especially if we look closer at the scatter plot and see that the points appear to form a few distinct clumps. Is it possible we missed something important? Let's break the data down by species by adding the argument `color = "species"`. This will color each point by its species and draw separate regression lines for each species.

```
ggscatter(data = penguin, x = "bill.length.mm", y = "bill.depth.mm", color = "species",
          add = "reg.line",
          title = "Bill Length and Depth by Species")
```



Once we control for species, we see the relationship between bill length and depth is actually positive! The clumping we saw earlier was penguin species. If we look at the body mass histogram we made earlier, this makes sense. the distribution of weight was quite different for each species. If the penguins are different sizes, that could play a role in their bill measurements.

This scenario where a trend will reverse when grouping is introduced is known as *Simpson's Paradox*. Not every dataset will behave like this but it brings up an important conversation about truthfulness and helpfulness.

Both graphs are factually correct but only one of them is helpful and informative. When working with data, we need to be mindful that our insights are both truthful and allows the audience to make an informed decision.