Section 1

Alex Stephenson

Contents

Navigating RStudio

When you open RStudio, there are four panes by default. In the top left corner is the "Script" pane. Typing everything into the console is a pain and also error-prone. To avoid this, it's is helpful to make a script. In RStudio, you can create a new RScript by going to File -> New File -> RScript. Alternatively, you can use the keyboard shortcuts Cmd + Shift + N (Ctrl + Shift + N on Windows). There is a large payoff for becoming familiar with keyboard shortcuts.

To run code that you type in a Script, you can highlight the line with your cursor and click the "Run" button. Alternatively, you can place your cursor at the beginning of a line and click the "Run" button. Finally, you can move the cursor to the line of interest and press Cmd + Enter (Ctrl + Enter on Windows).

In the bottom left pane is the Console. The console is where the code we write is executed. You can type code directly into the console.

In the top right corner is the Environment pane. All the variables and functions that we write will be saved here. In the bottom right pane is a set of miscellaneous tabs that give us access to our file system, any plots/graphs we make, the packages on our system, and R's help.

Unquestionably the best help manual for R is located at https://google.com.

Packages

An R package is a "collection of functions, data, and documentation that extends the capabilities of base R" (Wickham & Grolemund 2020). Almost all the packages that we use in this class are part of the "tidyverse." **This is on purpose.** These packages share a common philosophy of data and programming and so work together well.

While I cannot guarantee that your future job will use R[1], based on developer surveys, if it does, developers likely use at least some part of the tidyverse.

There are lots of great packages for R that are outside of the tidyverse. I encourage you to explore other packages as the semester moves on. The first package outside of the tidyverse that we will install is the `palmerpenguins` package. This is partly because penguins are cute and partly because we need a dataset that isn't historically exclusionary.

[1]In general, it will be Excel for non-research jobs

Installing Packages from CRAN

There are two ways to install packages in RStudio.

Point and Click

- Packages Panes -> Click the Install Button -> Type the package (or packages) name you want to install. For this class, we need tidyverse and palmerpenguins.

Code In the console, type and run the following line.

```r
# This is a comment. R ignores everything we type after
# the #. Comments are good. You should use them to tell people # what your code is doing and why.
install.packages(c("tidyverse", "palmerpenguins"))
```

You only have to install a package once, not every time you open RStudio.

Install Packages from other Sources

Sometimes this semester or in the future, you may be interested in a package not yet on CRAN.

To install a package from another source, use the `remotes` package. Here is an example of installing a package from GitHub.

```r
## The :: is a way to reference functions from a package
## without explicitly calling library(PACKAGE_NAME)
remotes::install_github("asteves/tayloRswift")
```

Loading Packages

Load packages at the top of your script with library().

Load the packages you need before writing any code. You cannot use any functions, objects, or help files in a package until you load the package with `library()`.

```r
library(palmerpenguins)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

When you run these lines of code, your console will output some text. This tells you which packages are being loaded by tidyverse and any functions from the tidyverse that conflict with functions in base R. This is fine and not something to worry about.

Data Frames

The `palmerpenguins` package gives us access to the penguins data frame, a kind of **data structure**. A data frame is a rectangular collection of variables and observations. In the penguins dataset, there are 334 observations (rows) and 8 variables (columns).

Let's practice two patterns in R: creating a variable and variable assignment. We are going to create a variable called df that is assigned the penguins dataset.

The general template for variable assignment is:

```
NAME_OF_VARIABLE <- THING_YOU_ASSIGN_TO_IT
```

```
df <- penguins

# Note that this also works. Pick your favorite and be consistent
df = penguins
```

We can view the result of a variable assignment in the console by simply typing it in and executing.

```
df
```

```
## # A tibble: 344 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

Here are some other examples

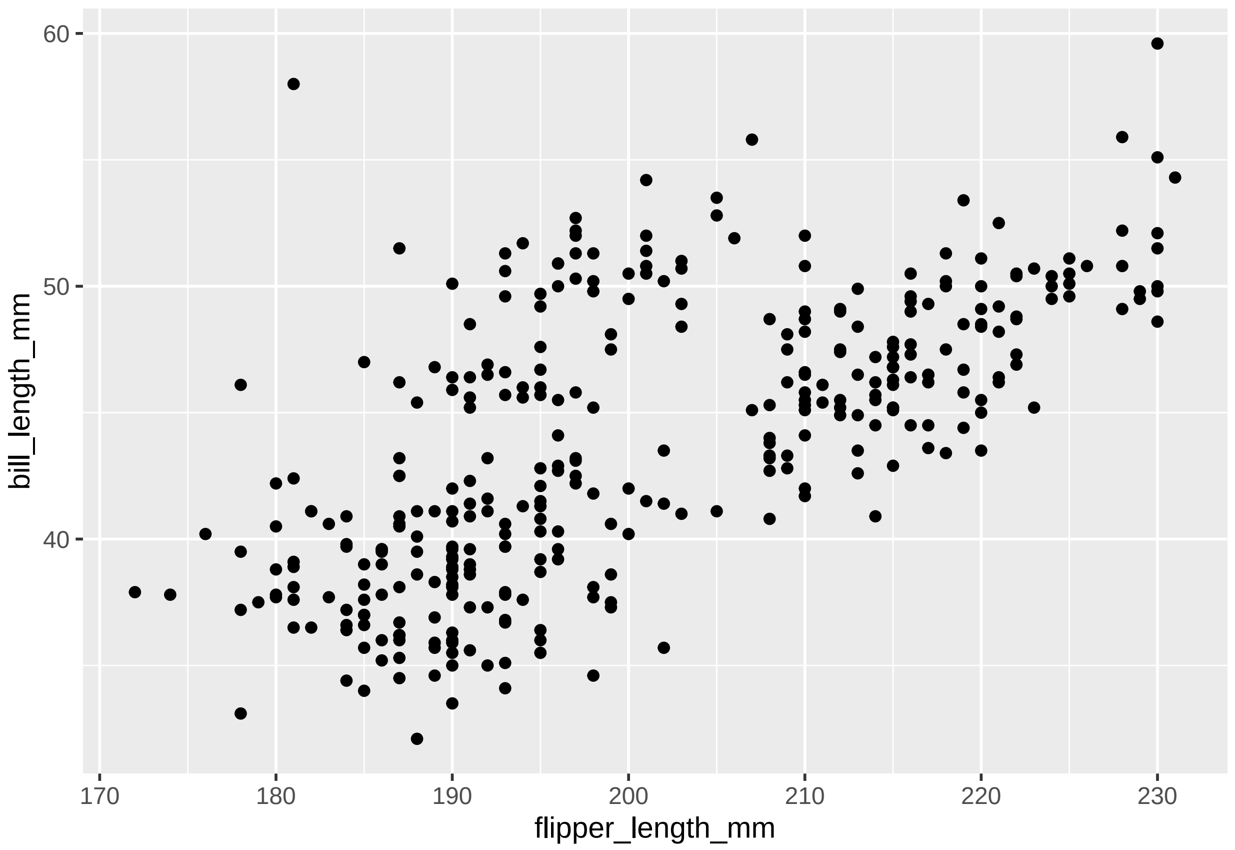
```
# Assign a number to a variable
five <- 5

# Assign a string to a variable
class <- "PS133"

# Assign the result of a computation to a variable
add2to5 <- 2 + 5

# Assign a vector to a variable
vector <- c(five, add2to5)
```

Visualize Data

We will make all of our graphs with the `ggplot2` package. `ggplot2` is one of the packages loaded when you call `library(tidyverse)`.

Our first graph is going to be a scatterplot of penguin bill length by flipper length.

```
# Note that you could type everything as one line
# but it doesn't look readable. R doesn't care, so write code
# for humans
ggplot(data = df) +
  geom_point(mapping = aes(x = flipper_length_mm, y = bill_length_mm))
```



What's happening here? To make a plot, we begin with the function '`ggplot`. This function creates the base layer of our graph. The first **argument** to `ggplot` is the dataset we want to use in the graph.

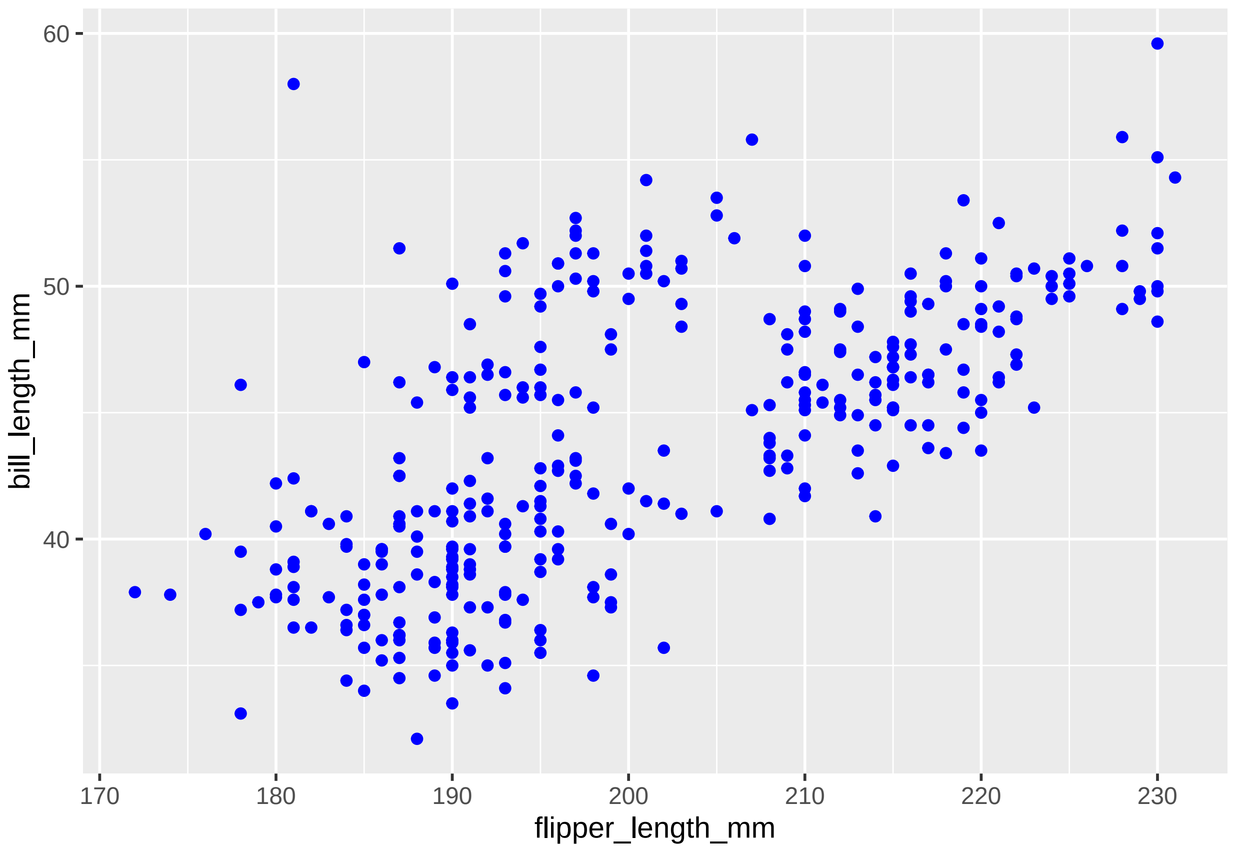
This is followed by a `+` sign, which tells the `ggplot` function that you want to chain another layer to the graph. The function `geom_point` adds a layer of points to the graph. `geom_point` takes an argument `mapping`, which is always paired with the aes() function. `aes()` is short for aesthetics. You need to specify the x-axis variable and the y axis variable from your dataset.

More generally, we can think of visualization with the following workflow template:

```
ggplot(data = <DATA FRAME>)+
  <geom_function>(mapping = aes(x = X_VARIABLE, y = Y_VARIABLE, ...), ...)
```

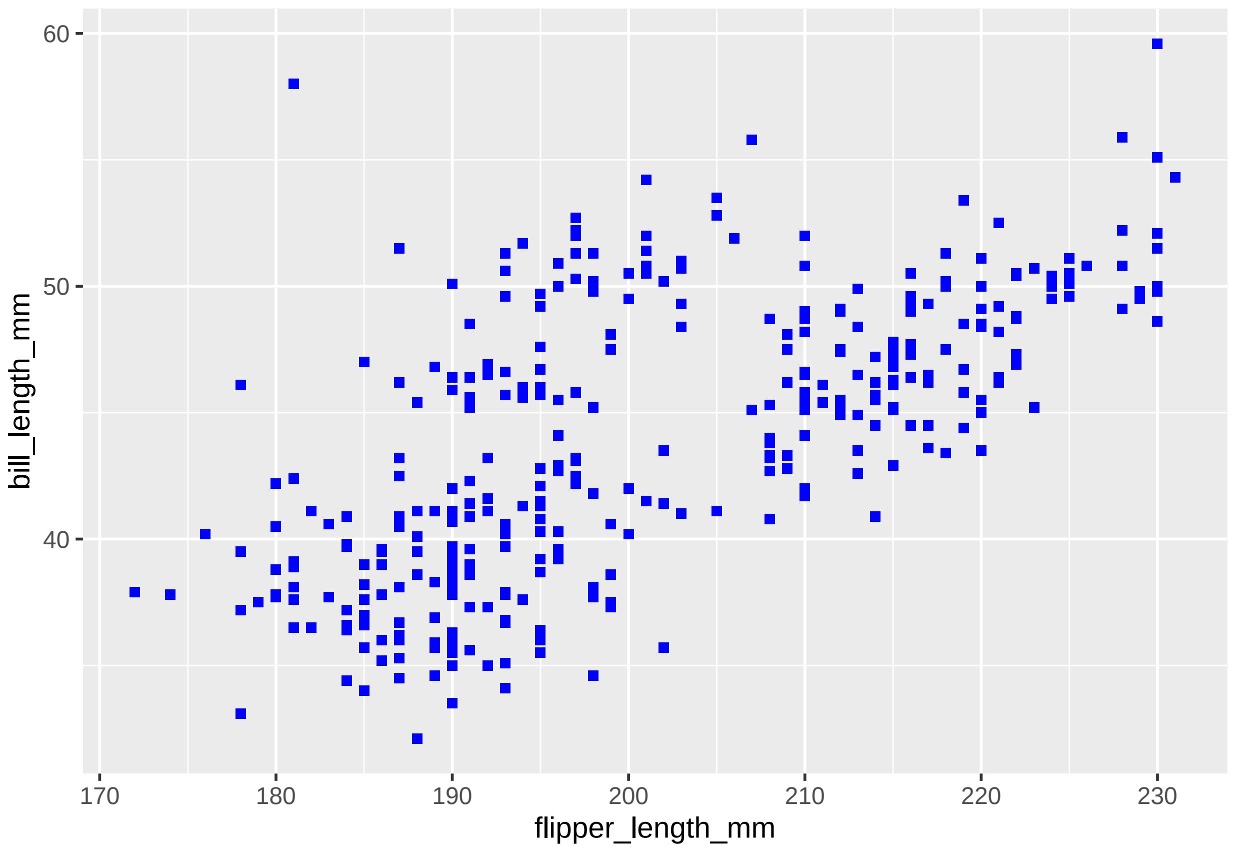
Here is an example of using other aesthetics arguments to change the plot, so the points are blue.

```
ggplot(data = df)+
  geom_point(mapping = aes(x = flipper_length_mm, y = bill_length_mm), color = "blue")
```



Note that we have defined color outside the `aes()` function because we manually set its property. We can do the same thing to change the points. Suppose we want to use squares instead of circles because we like right angles.

```
ggplot(data = df)+
  # The shape parameter controls the shape
  # It happens to be the case that 15 is square
  geom_point(mapping = aes(x = flipper_length_mm, y = bill_length_mm), color = "blue", shape = 15)
```



Putting it all together

Once you feel comfortable with how to make this graph, you have almost[2] all the foundational knowledge you need to solve every assignment in this class. Here are all the things we have seen so far:

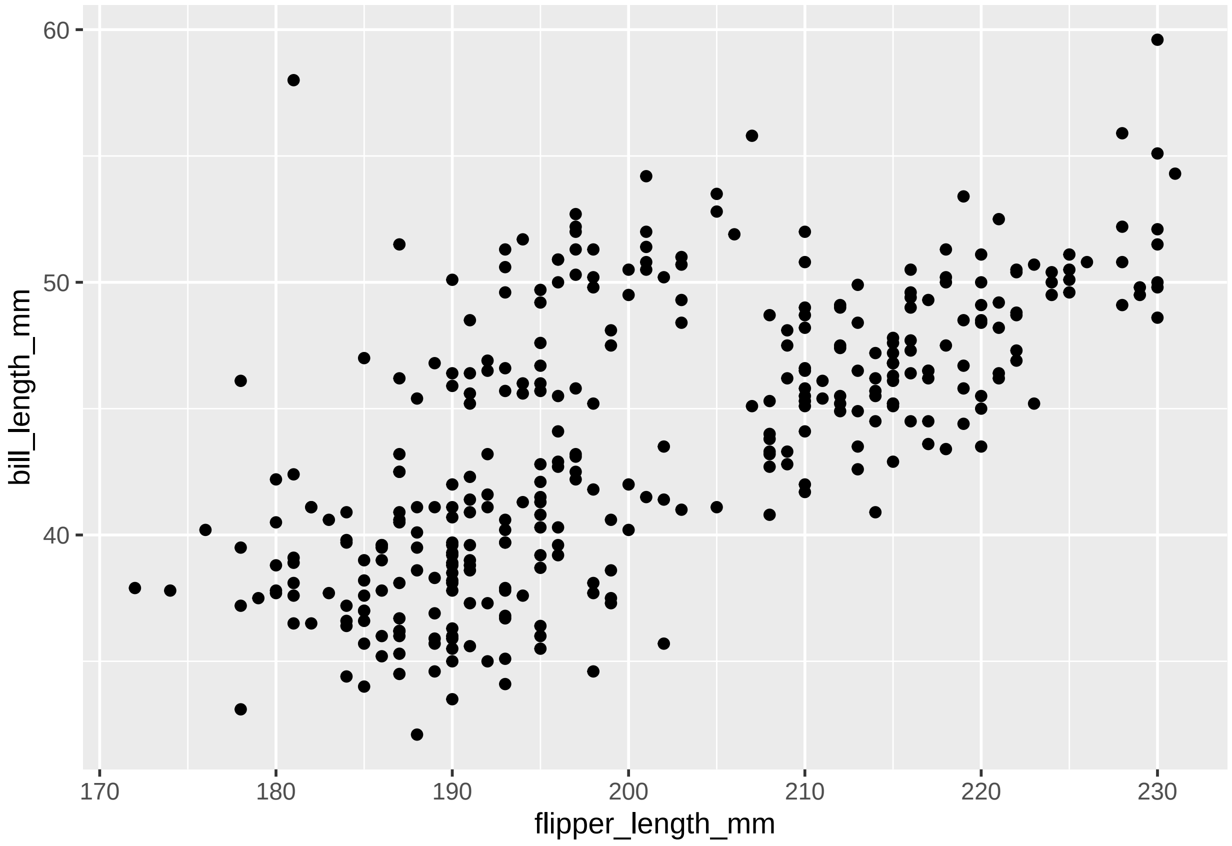
- Installing packages: Gets us useful functions
- Loading packages: This allows us to use those packages in our work
- Looking at a data frame: This allows us to get an organized dataset
- Functions: "recipes" to do something
- Arguments: the "ingredients" the function needs to do its job
- Variable assignment: The way to store values to use again
- Graphics: Making pretty pictures.

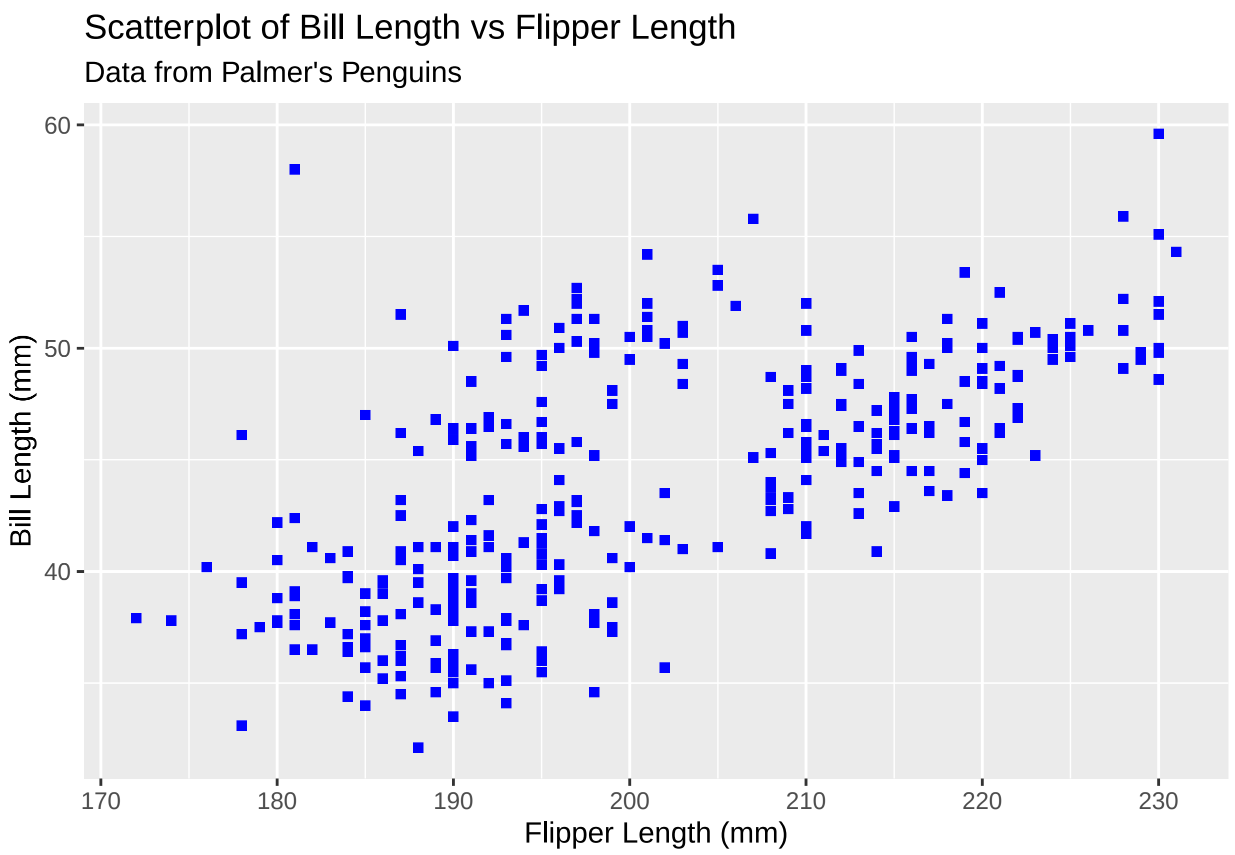
Here is the equivalent program of everything we have typed so far at once.

```r
## Load packages we need
library(palmerpenguins)
library(tidyverse)

## Assign a variable df to the penguins dataset
df <- penguins

## Make a scatterplot
ggplot(data = df) +
```

[2]We'll get there.

```r
  geom_point(mapping = aes(x = flipper_length_mm, y = bill_length_mm))

## Make another scatterplot
## This time, make the points blue squares

ggplot(data = df)+
  # The shape parameter controls the shape
  # It happens to be the case that 15 is square
  geom_point(mapping = aes(x = flipper_length_mm, y = bill_length_mm), color = "blue", shape = 15)+
  # Here are some additional aspects of the plot
  # Examine for yourself what changes when adding
  # the following lines
  xlab("Flipper Length (mm)")+
  ylab("Bill Length (mm)")+
  ggtitle("Scatterplot of Bill Length vs Flipper Length",
          subtitle = "Data from Palmer's Penguins")
```




Scatterplot of Bill Length vs Flipper Length
Data from Palmer's Penguins
Bill Length (mm)
60
50
40
170
180
190
200
210
220
230
Flipper Length (mm)