Plotting Sample Data using ggplot

Nick Sumpter (Edited by Eddie-Williams Owiredu & Guy Twa)

2023-08-25

Today's Lab

In today's lab, we will learn to plot data from the `iris` dataset that was introduced last lab. To do this, we will use one of the many packages of the `tidyverse` known as `ggplot2`. The 'gg' stands for 'Grammar of Graphics' and this package enables highly customizable, publication-quality figures to be plotted in R.

Loading Packages and Data

To start with, let's load the `tidyverse` package that we installed last week, which includes the `ggplot2` package. As a reminder, installing a package merely makes it available on your device, you must load the package in each R session (using the `library` function) in order to use its functions. Additionally, we can load the `iris` data set into our Environment.

```
library(tidyverse)

iris <- iris
```

Workflow

1. The `ggplot()` function

Now that we have loaded both our data and our functions into our RStudio session, it is time to start plotting. Before we can do that, however, it is important that we go over a couple of the details of the `ggplot2` package and its main functions. The first function is `ggplot`. Try running the command `ggplot()`, including the parentheses. As you will see, this opened up the Plots panel in the lower right corner of your interface, but it only has the white background currently. Essentially, you have just told R "I am about to give you some information that I want plotted" and it responded by opening the space for it to be plotted.

```
ggplot()
```

Let's try searching for some help on the `ggplot` function using the Help panel in the bottom right corner of your interface. Click the panel then type 'ggplot' into the search bar and hit `Enter` / `Return` . Now, there is a lot of information to dissect here, but for right now lets just focus on the 'Arguments' section. As you can see, the Help panel lists all possible arguments that you can provide to the function, with a brief description of each one. You can ignore everything but the first sentence of the description for `data` and `mapping` . These sentences tell us that `data` supplies the function with the default dataset for the plot, and `mapping` provides the aesthetic mappings for the plot.

2. The `data` argument

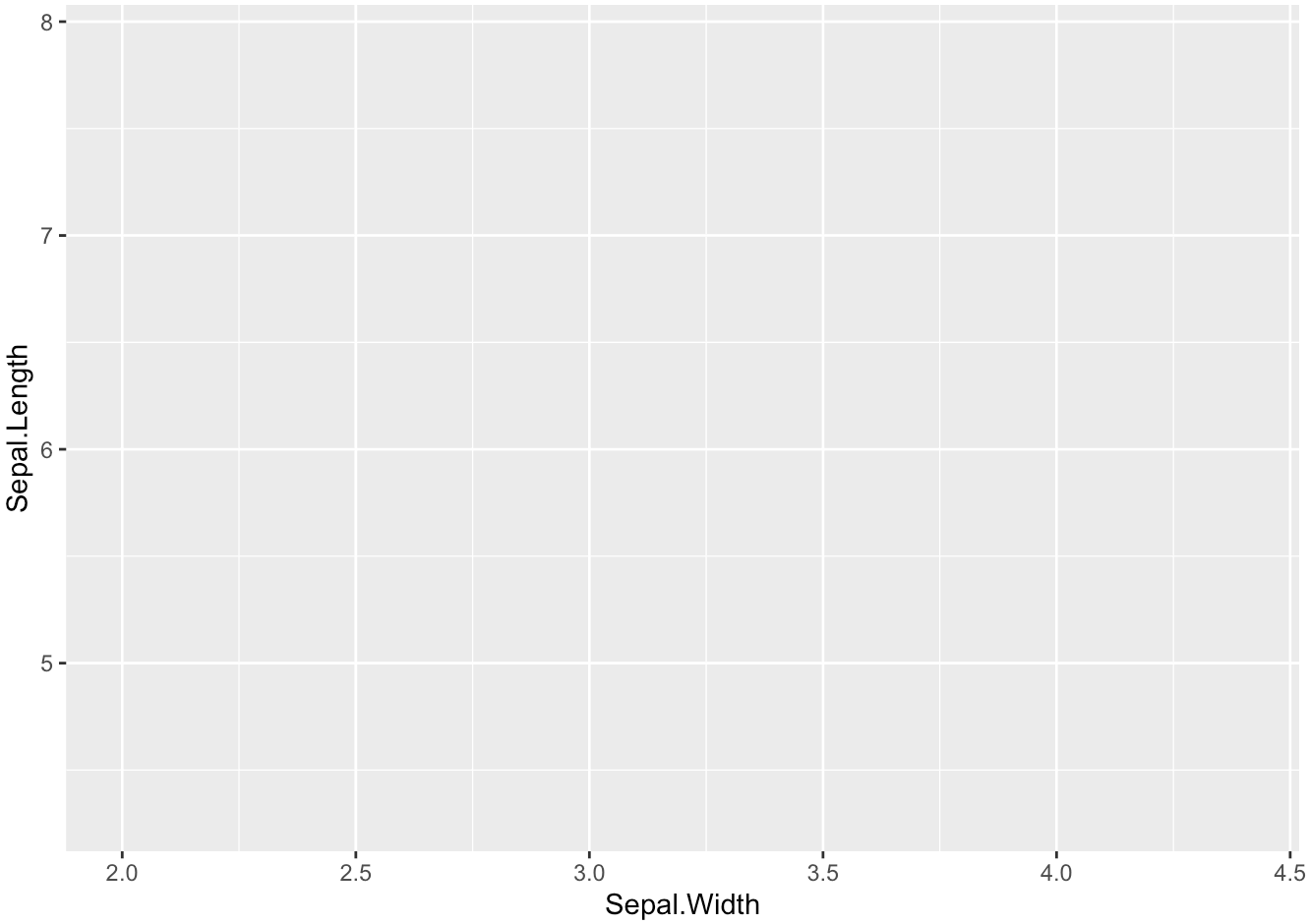
Now, we need to give `ggplot` some data to use for plotting. We can do this using the `data` argument of the `ggplot` function. Let's provide `ggplot` with our dataset `iris` . The 'Usage' section of the Help panel gives us a hint as to how to format our function.

```
ggplot(data = iris)
```

3. The `mapping` argument

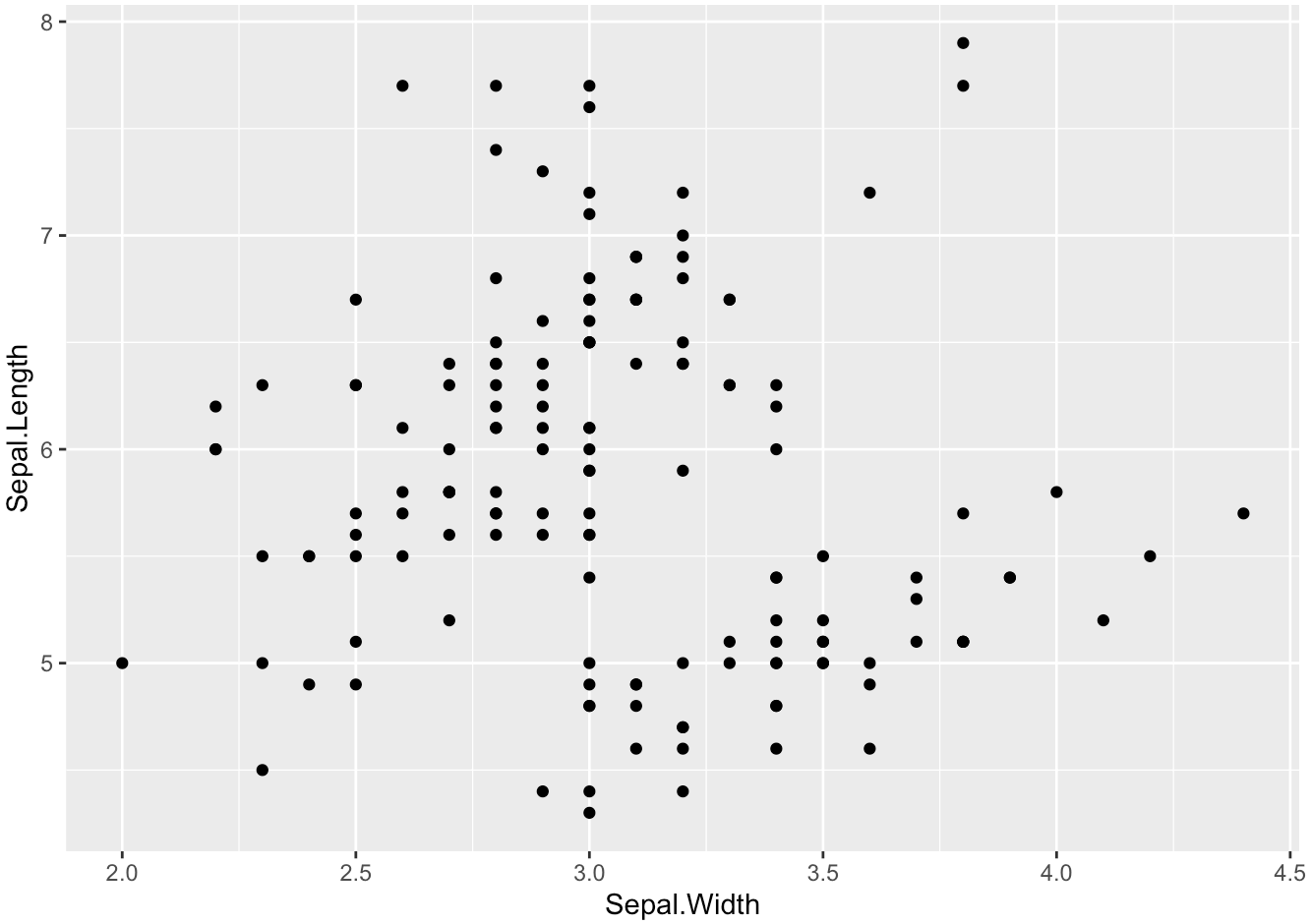
As you can see above, nothing has happened yet, and that is because we haven't told `ggplot` what to do with the data we provided. That is where the `mapping` comes in. This is essentially how you tell `ggplot` which column relates to the x-axis and which relates to the y-axis (plus some other things like color). So let's plot `Sepal.Length` on the y-axis and `Sepal.Width` on the x-axis. To do this, we need to supply it within another function called `aes` which stands for aesthetics (this is shown in the Usage section of the Help panel). We do this as follows:

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length))
```



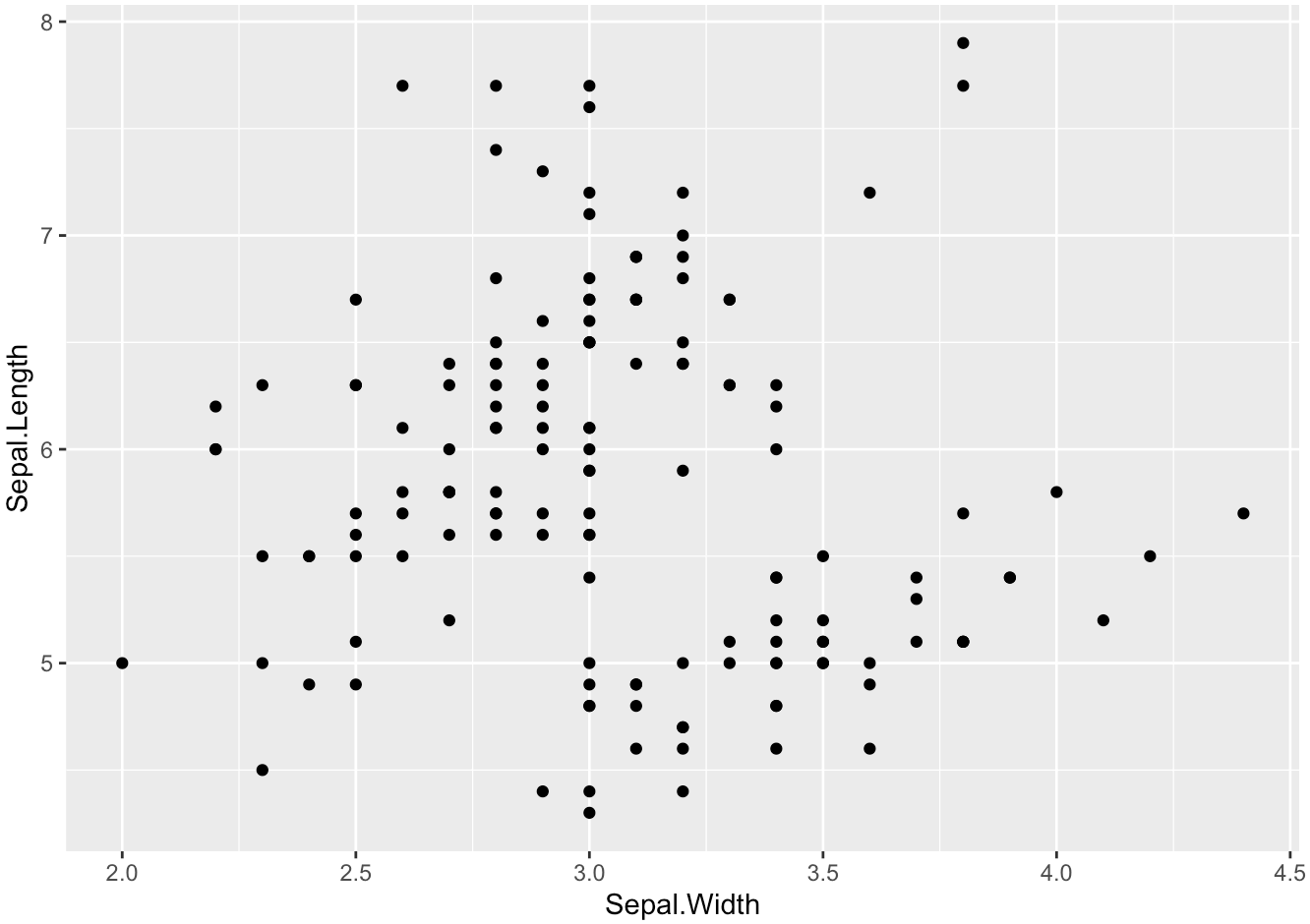
4. Right, so we are getting very close to a plot now, we have told R that we want a plot (using the `ggplot` function) that uses the `iris` dataset (the `data` argument) and maps `Sepal.Width` to the x-axis and `Sepal.Length` to the y-axis. As our next layer of the plot, lets plot all 150 pairs of Sepal values as a scatter plot using the `geom_point` function. To do this, we will add the `geom_point` function directly after the `ggplot` function using the `+` symbol.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) + geom_point()
```



Note: Throughout the remainder of the labs, I will write the `ggplot` code a little differently to above. Instead of having everything on one line, it is a lot easier to read the code by introducing each function on it's own line, like so:

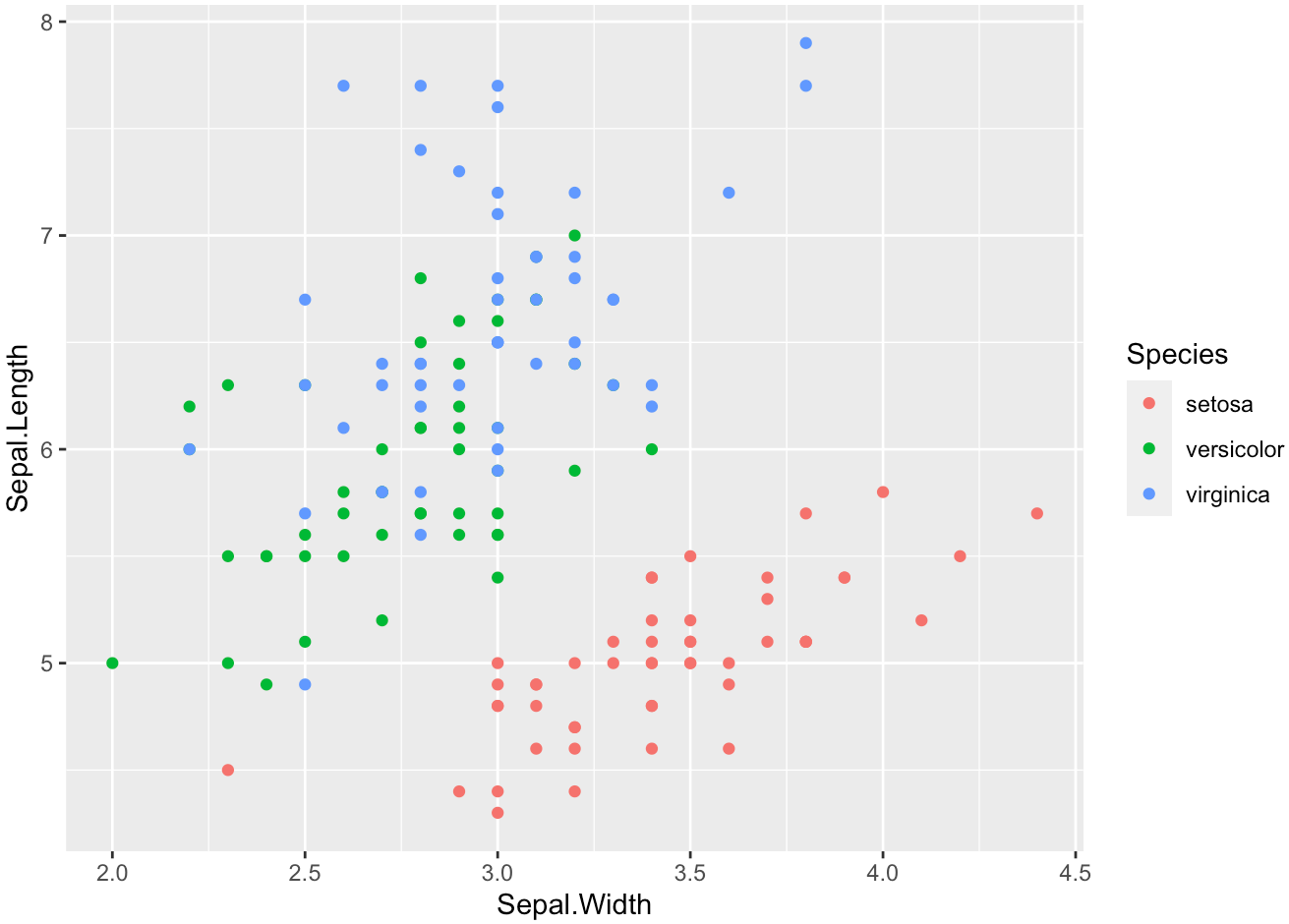
```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
```



This is exactly the same overall function, but we can now see each individual function as we apply it to the data. Note that you must end the line of code with a `+` in order for this to work correctly when running the `ggplot` function. Ending a line with a `+` will cause R to look to the next line for the remainder of the code. This is a really good habit to get into early on as it will make it much easier to look back at your code and understand it.

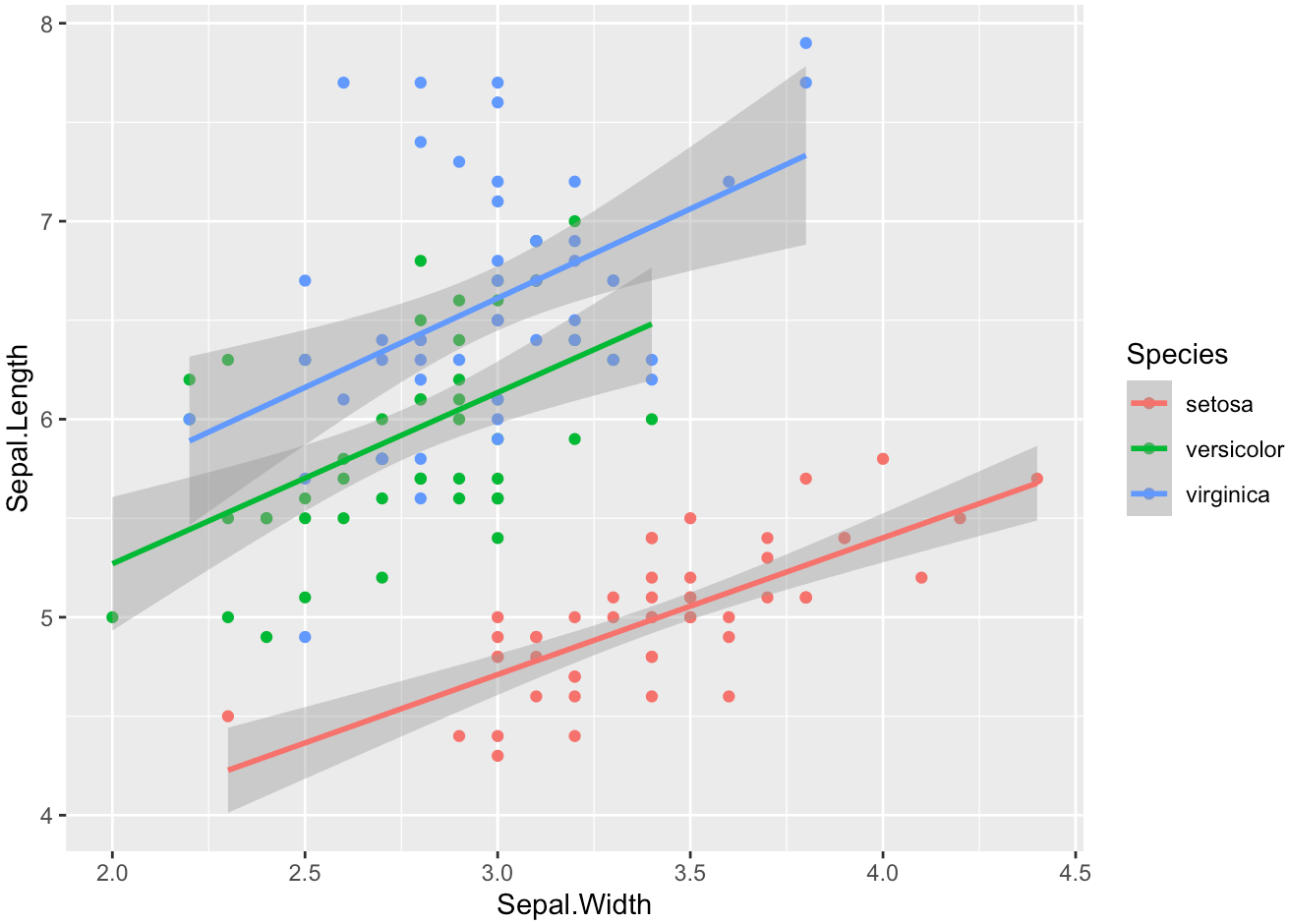
5. Great, we now have a scatter plot, but it looks a little messy. Perhaps if we colored the data points based on which species of flower they were from, it would make more sense. Lets do this using the `color` aesthetic in the `mapping` argument of `ggplot`.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point()
```



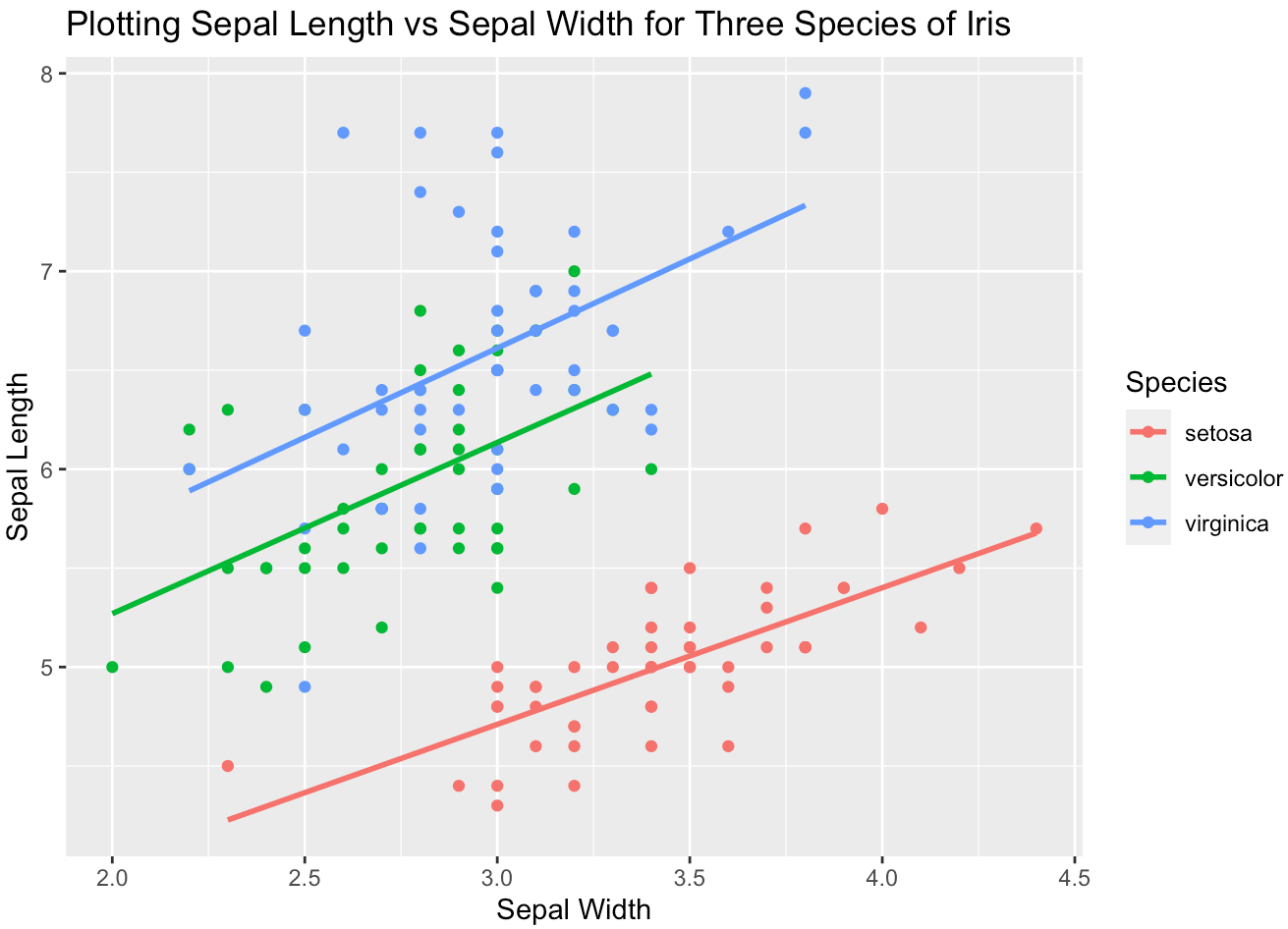
6. There we go, clearly the 'setosa' species of iris differs from the other two and therefore made the overall distribution of points look strange. Now if we wanted to visualize the relationship between `Sepal.Width` and `Sepal.Length` better, perhaps we could add a linear model for each species. Now you will start to see the real power of `ggplot`, and it comes in the form of layering plots on top of other plots. We have so far just given the `geom_point()` function to `ggplot` to tell it how to plot the data, lets try adding a new geom (called `geom_smooth`) that will generate a line that plots the relationship between variables. By default this will be a complicated model called a spline, to make it a simple linear model we will need to add a `method` argument to `geom_smooth`. Try searching `geom_smooth` in the Help panel and see which other options are available for the `method` argument.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  geom_smooth(method = "lm")
```



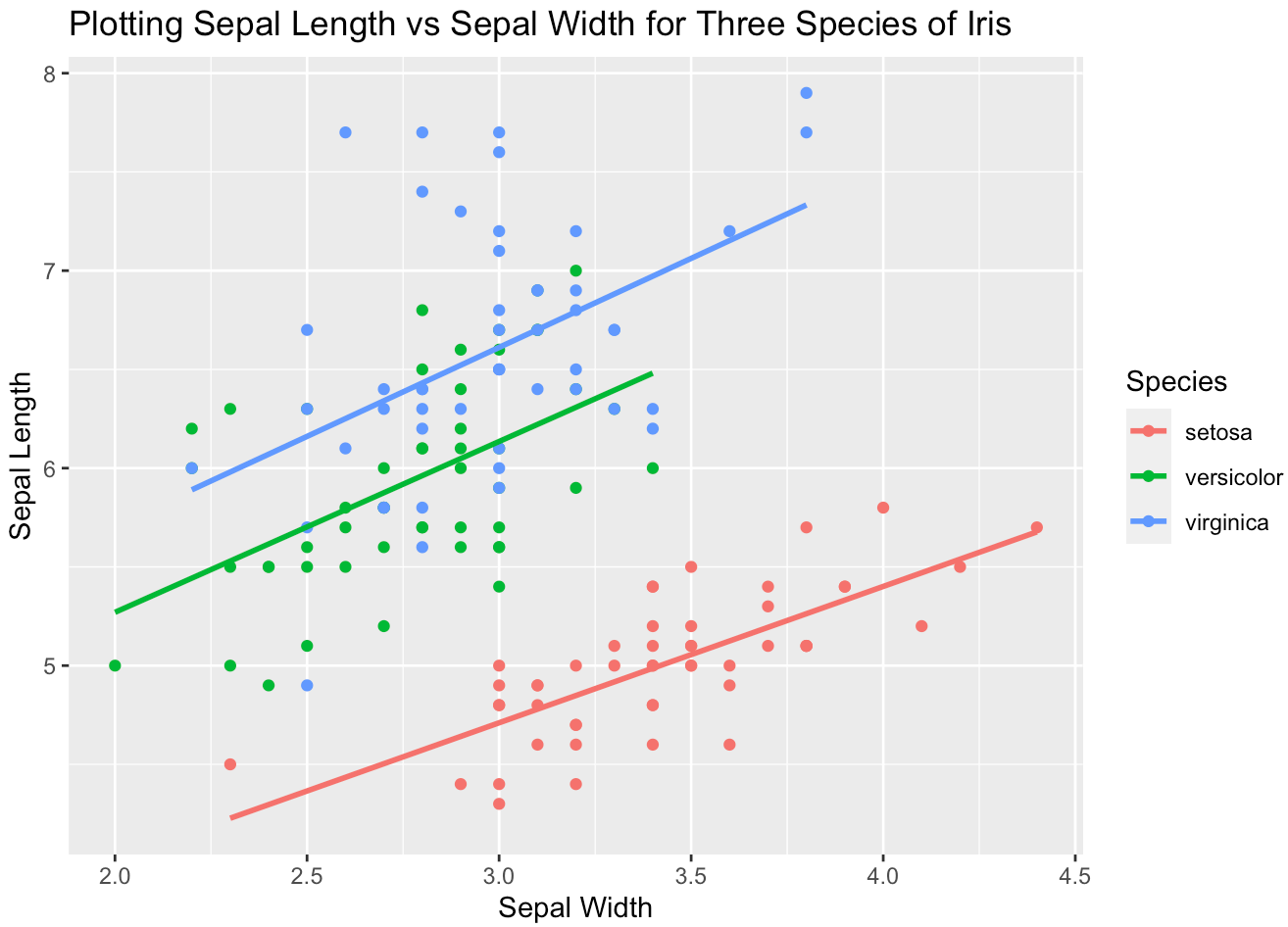
7. Finally, let's try to clean up the plot a little by changing the axis labels and adding a title. These can be added with the `labs` function. Additionally, lets remove the gray shading on the `geom_smooth` layer by adding the `se = FALSE` argument.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Sepal Width", y = "Sepal Length", title = "Plotting Sepal Length vs Sepal Wi
dth for Three Species of Iris")
```



Note: Just like how we can write each function on a new line as long as we end the line of code with a `+`, we can also do the same by ending the line of code with a `,`. This can be nice for writing code that is easier to read, but should only really be used when the line of code is hard to read on one line. For the above example, it will be a good idea to introduce new lines within the `labs` function like so:

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Sepal Width",
       y = "Sepal Length",
       title = "Plotting Sepal Length vs Sepal Width for Three Species of Iris")
```



Again, this makes no difference to how the code runs, it just makes it easier to read. Making your code easy to read is an extremely good habit to get into early on.

Independent Practice

1. Plot the relationship between Petal Length and Petal Width in a similar way to how we plotted Sepal Length vs Sepal Width.
2. Try and modify the shape of the points based on which Species they belong to.