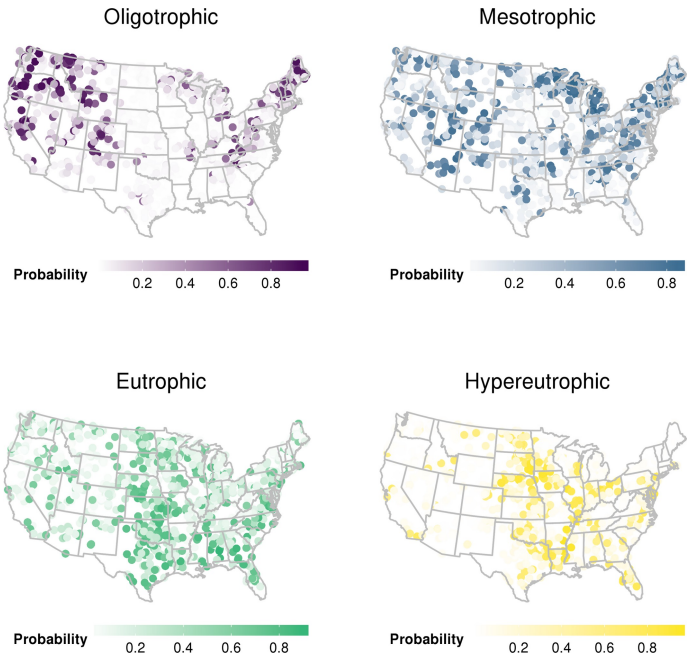


Figure 1: Trophic State Modeling Results

Data Visualization with ggplot2

Visualizing data is an area where R really shines. There are many ways to plot data with R and these include base R, `lattice`,`grid` and , `ggplot2`. The only one we will work with is `ggplot2`, which is now (I have no data to back this up), the de-facto standard for visualizing data in R. Given that `ggplot2` is general package for creating essentially ALL types of visualizations, it can seem quite complex (and it is). What I hope you will get out of this section is a basic understanding of how to create a figure and, most importantly, how to find help and examples that you can build off of for your own visualizations. If you want to read more about why some people choose base plotting vs `ggplot2`, the twitter/blogosphere "argument" between Jeff Leek and David Robinson is worth some time.

Lesson Outline

- Examples of greatness
- Basics of `ggplot2`
- Example explained

Exercise

- Exercise 4.1

Examples of what is possible

Before we get started, I do like to show what is possible. A couple of geospatial examples of maps created in R.

A few examples of maps built with R show this:

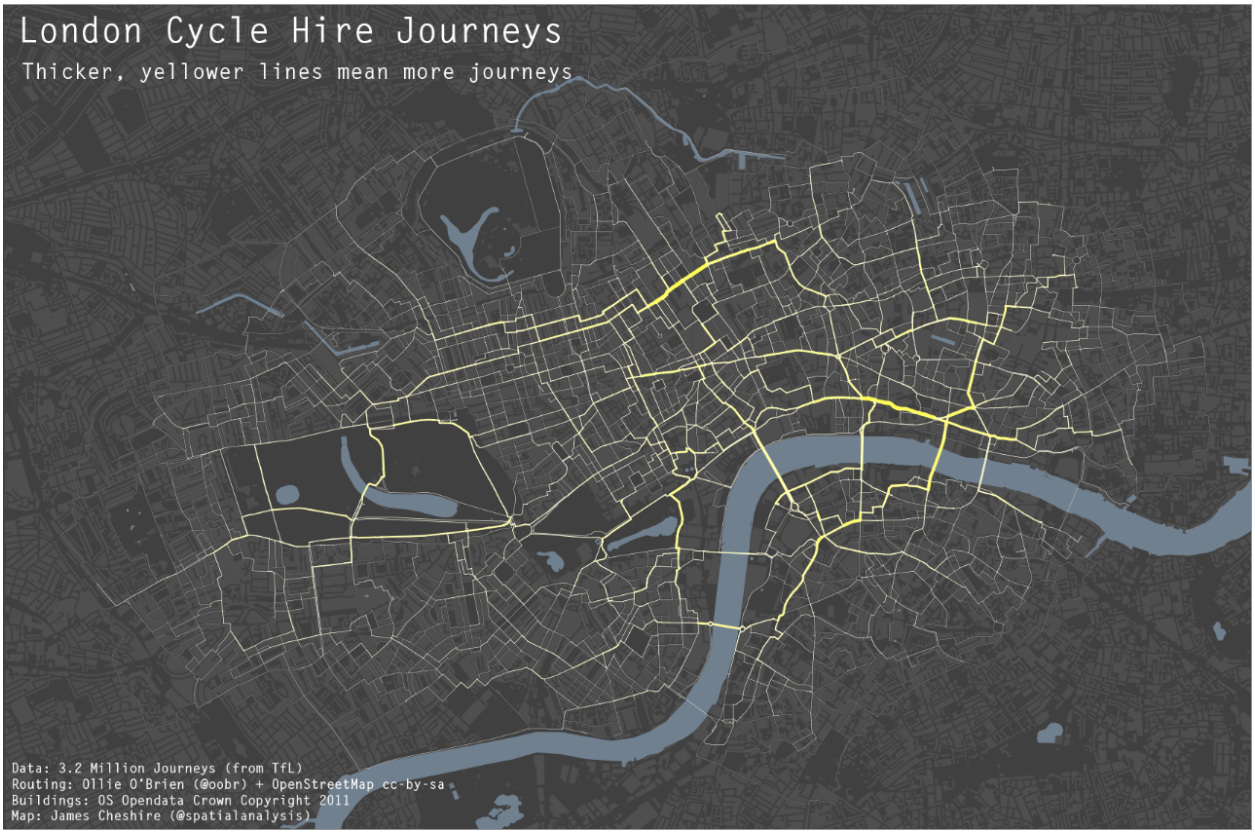
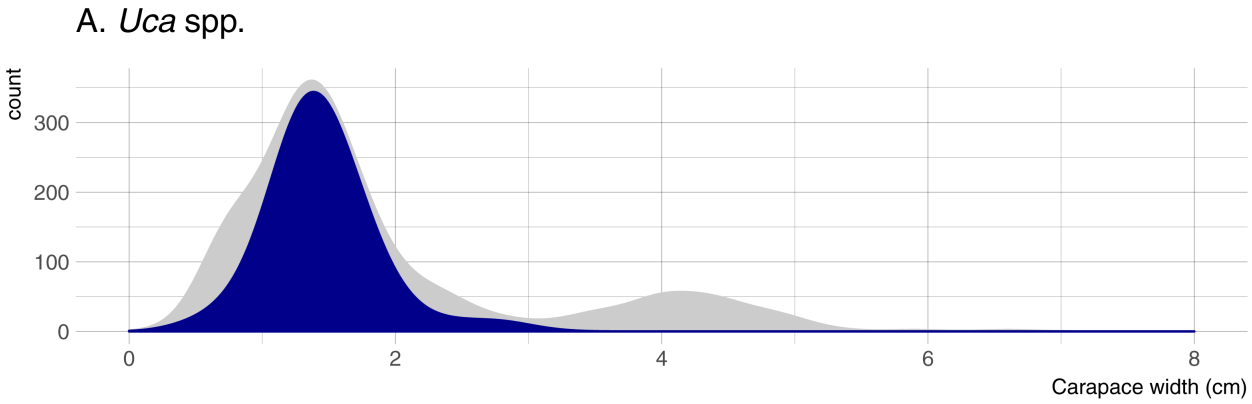
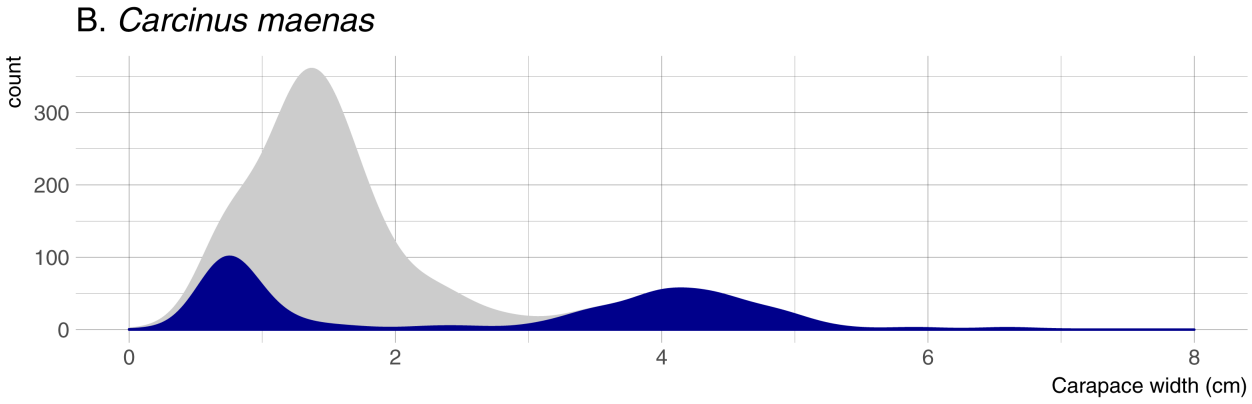


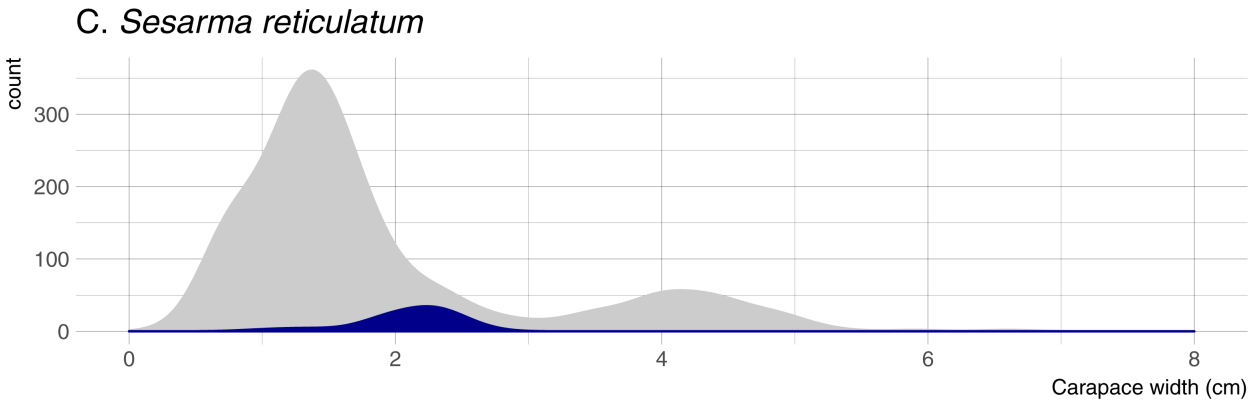
Figure 2: London Bike Hires

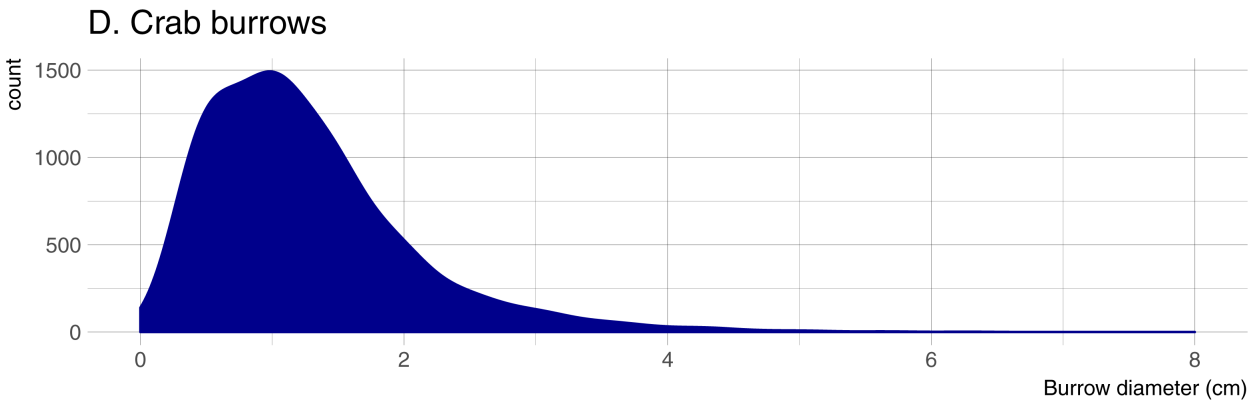
Figure 3: Facebook Users

More examples from Jeff's work









from: Raposa et al. (2018). Top-down and bottom-up controls on overabundant New England salt marsh crab populations. PeerJ. https://doi.org/10.7717/peerj.4876

from: Kuhn et al. (2018) Performance of national maps of watershed integrity at watershed scales. Water. https://doi.org/10.3390/w10050604

And some cool examples using `ggplot2` with `plotly`.

http://blog.revolutionanalytics.com/2014/11/3-d-plots-with-plotly.html

Lastly, so that you know that there are many (often cool) mistakes that lead up to a final visualization there is Accidental aRt. And for a specific example ...

And the map I showed earlier of the trophic state probability had as one of its early iterations this "psychadelic doughnut"

(**ht to Anne Kuhn, my office mate, for the name**)

A few other great links that I have recently found are also useful for inspiration. First, is a repository on GitHub that has most (all?) of the currently available color palletes in R: https://github.com/EmilHvitfeldt/r-color-palettes. Second, the R graph gallery is a fantastic resource for seeing all that is possible for visualization in R and the code on how to do it!!

Now that we are sufficiently motivated, lets take a step back to the very basics.

Introduction to ggplot2: scatterplot

When you first get a dataset and are just starting to explore it, you want do be able to quickly visualize different bits and pieces about the data. I tend to do this, initially, with base R. But since our time is short, we are going to focus our efforts just on `ggplot2`.

A lot has been written and discussed about `ggplot2`. In particular see here, here and here. The gist of all this, is that `ggplot2` is an implementation of something known as the "grammar of graphics." This separates the basic components of a graphic into distinct parts (e.g. like the parts of speech in a sentence). You add these parts together and get a figure.

Before we start developing some graphics, we need to do a bit of package maintenance. If `ggplot2` had not be installed (it should be by now), install it and make sure to load up the package with `library()`

```
install.packages("ggplot2")
library("ggplot2")
```

With that finished, we can now use `ggplot2`. First thing we need to do is to create our ggplot object. Everything will build off of this object. The bare minimum for this is the data (handily, `ggplot()` is expecting a data frame) and `aes()`, or the aesthetics layers. Oddly (at least to me), this is the main place you specify your x and y data values.

```
# aes() are the "aesthetics" mappings.  When you simply add the x and y
# that can seem a bit of a confusing term.  You also use aes() to
# change color, shape, size etc. of some items
iris_gg <- ggplot(iris,aes(x=Petal.Length,y=Petal.Width))
iris_gg
```

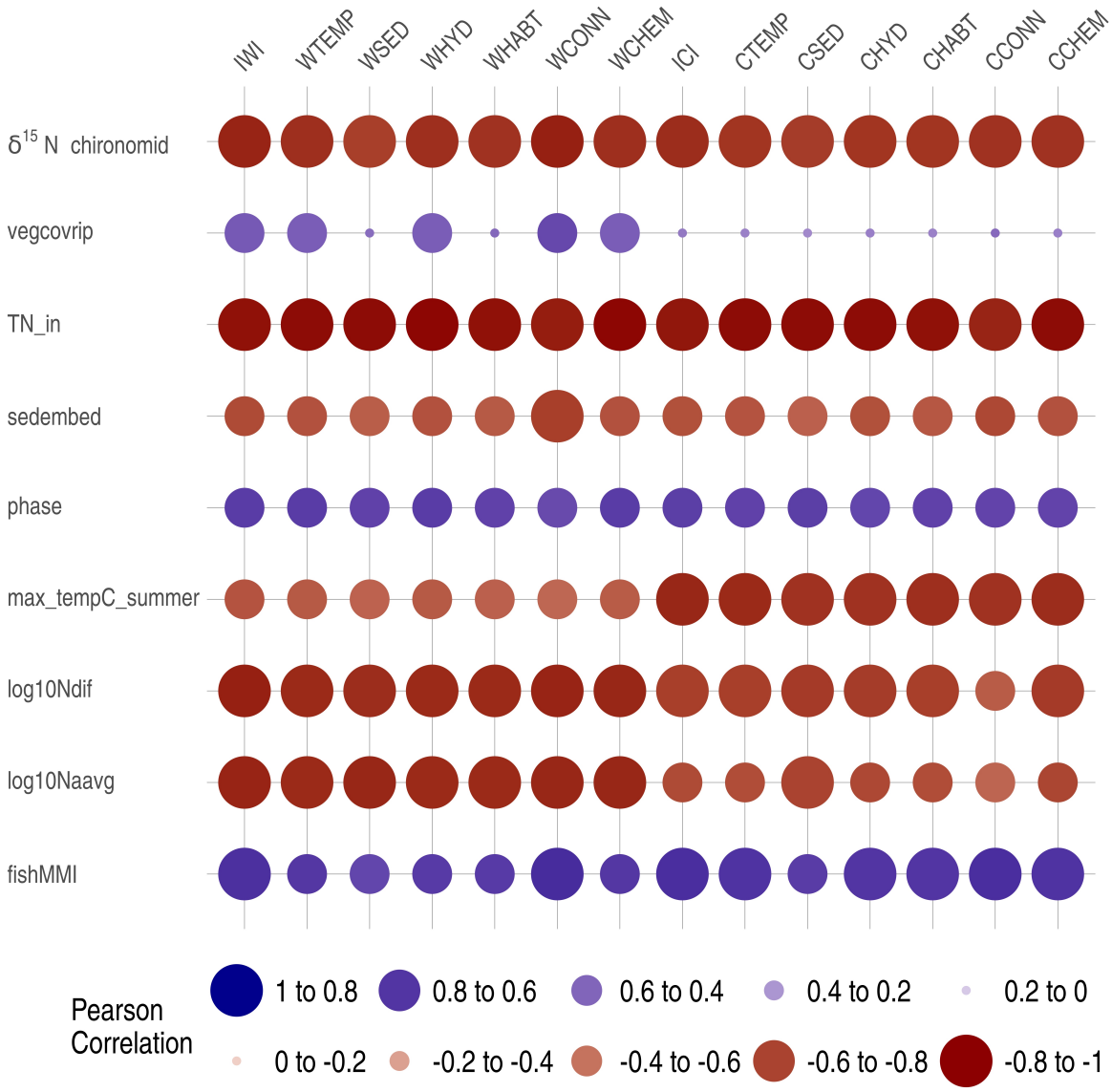


Figure 4: heatmaps

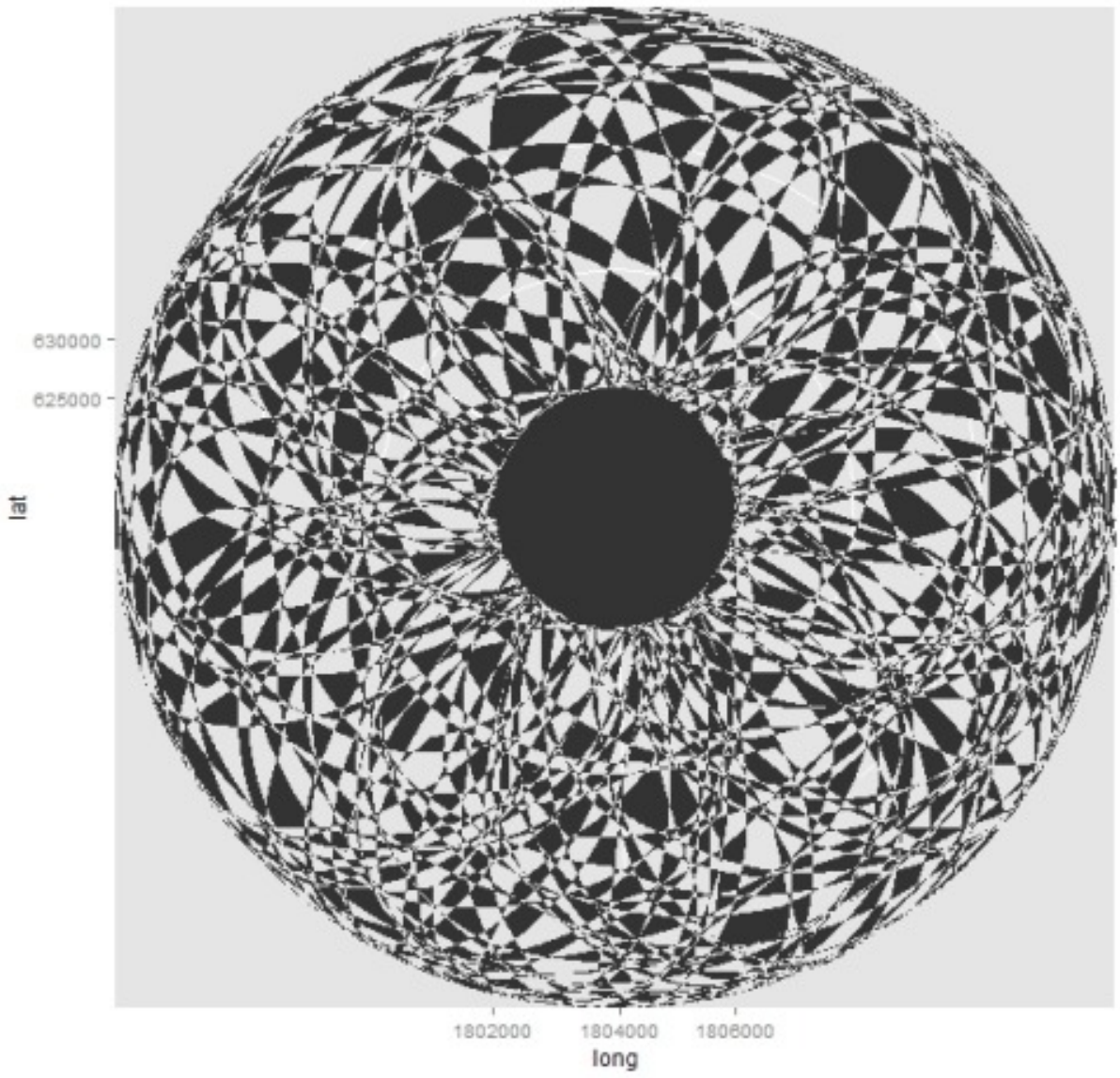
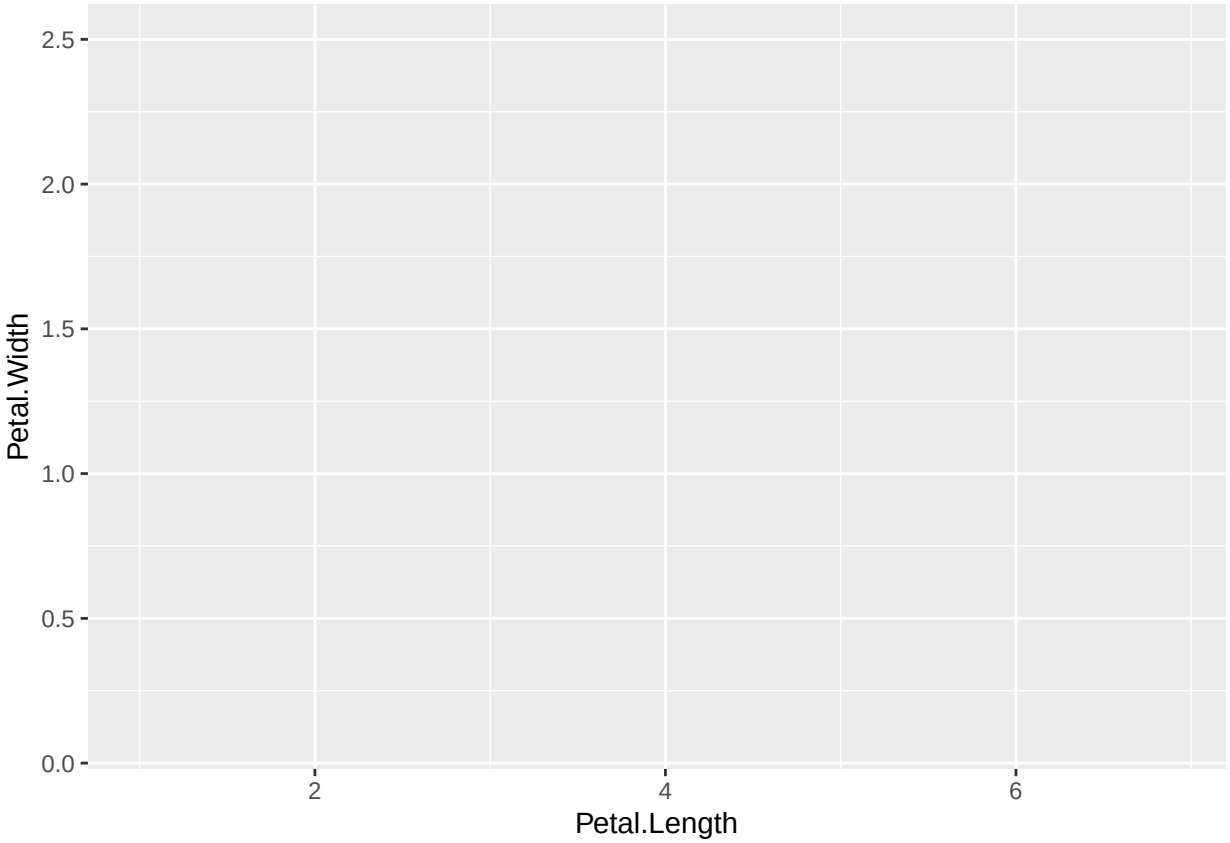


Figure 5: pd

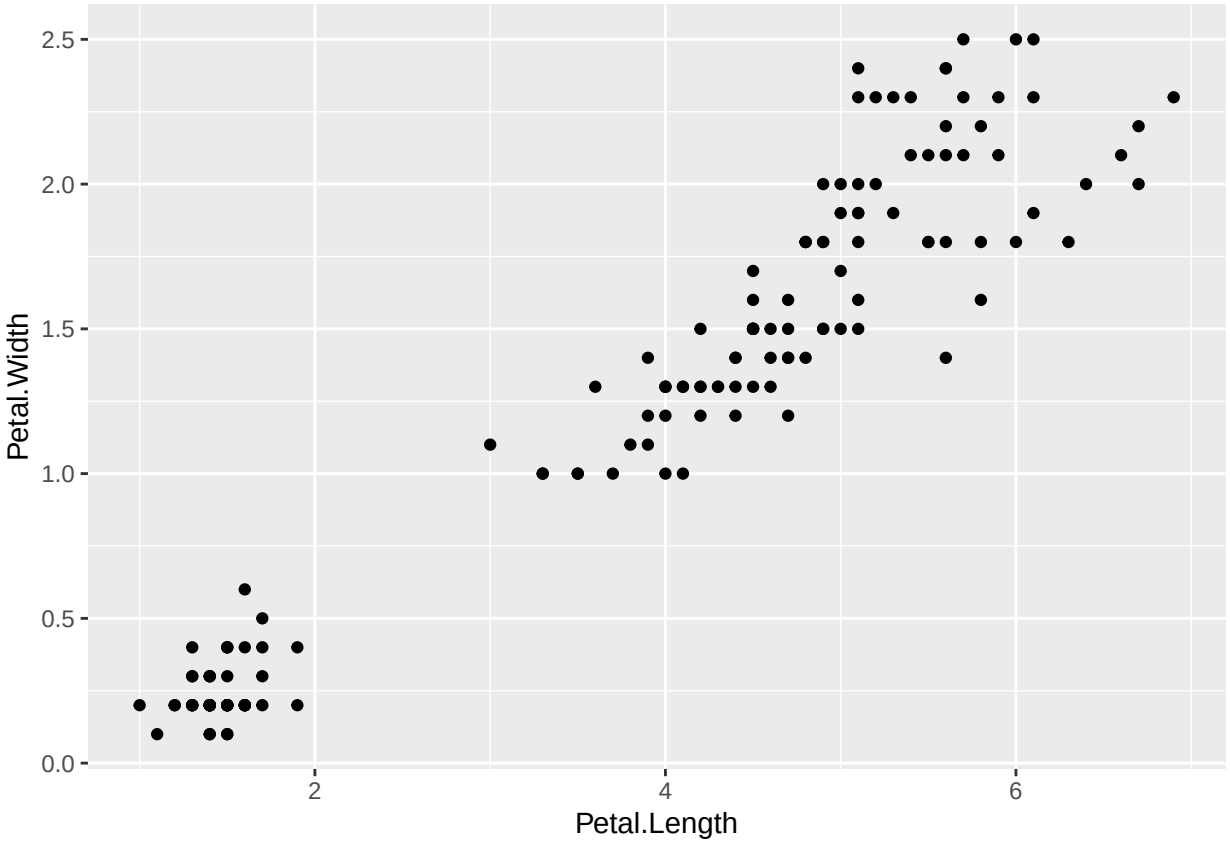


Great, nothing plotted... All we did at this point is create an object that contains our data and what we want on the x and y axes. We haven't said anything about what type of plot we want to make. That comes next with the use of geometries or `geom_`'s.

So if we want to simply plot points we can add that geometry to the ggplot object.

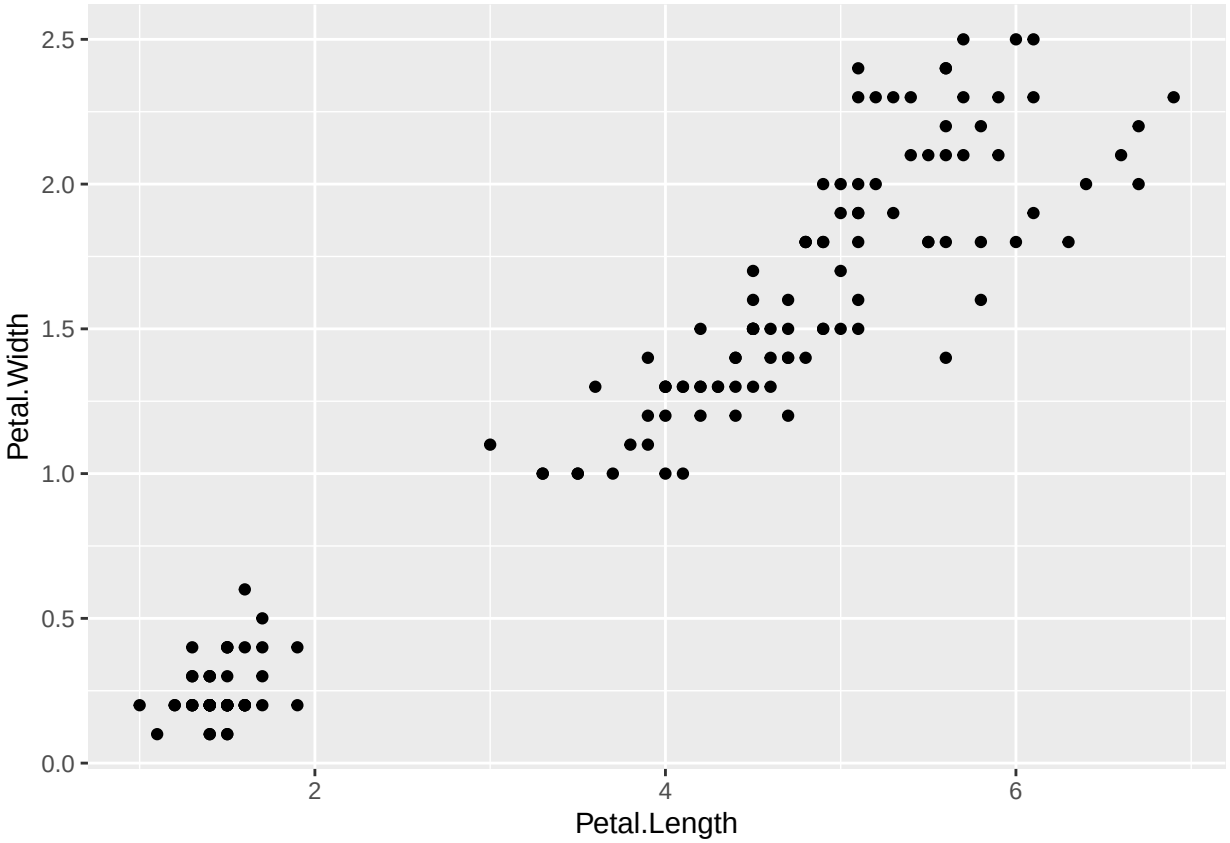
A side note on syntax. You will notice that we add new "things" to a ggplot object by adding new functions. In concept this is somewhat similar to the piping we talked about earlier. Essentially it takes the output from the first function as the input to the second. So to add points and create the plot, we would do:

```
#Different syntax than you are used to
iris_gg +
  geom_point()
```



It is usually preferrable to save this to an object.

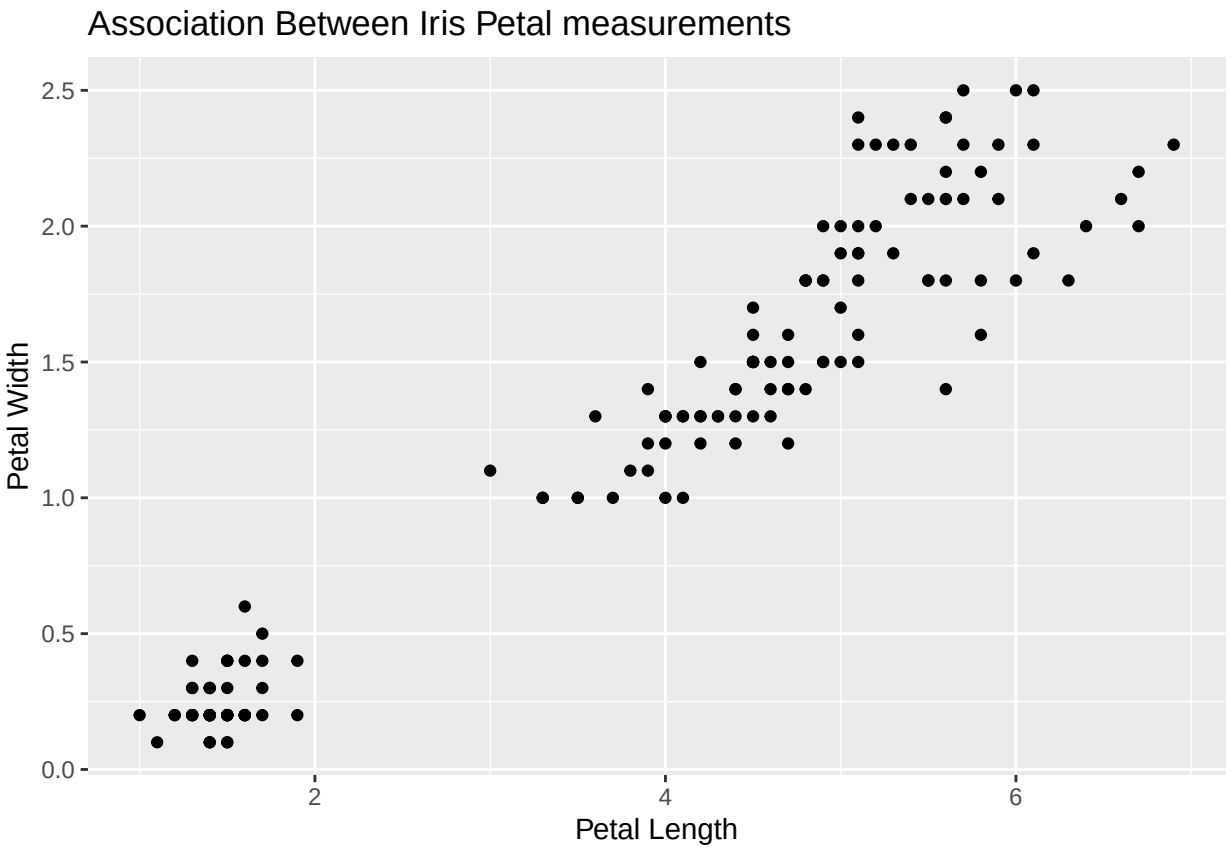
```
#This too can be saved to an object
iris_scatter <- iris_gg +
  geom_point()

#Call it to show the plot
iris_scatter
```



Not appreciably better than base, in my opinion. But what if we want to add some stuff...

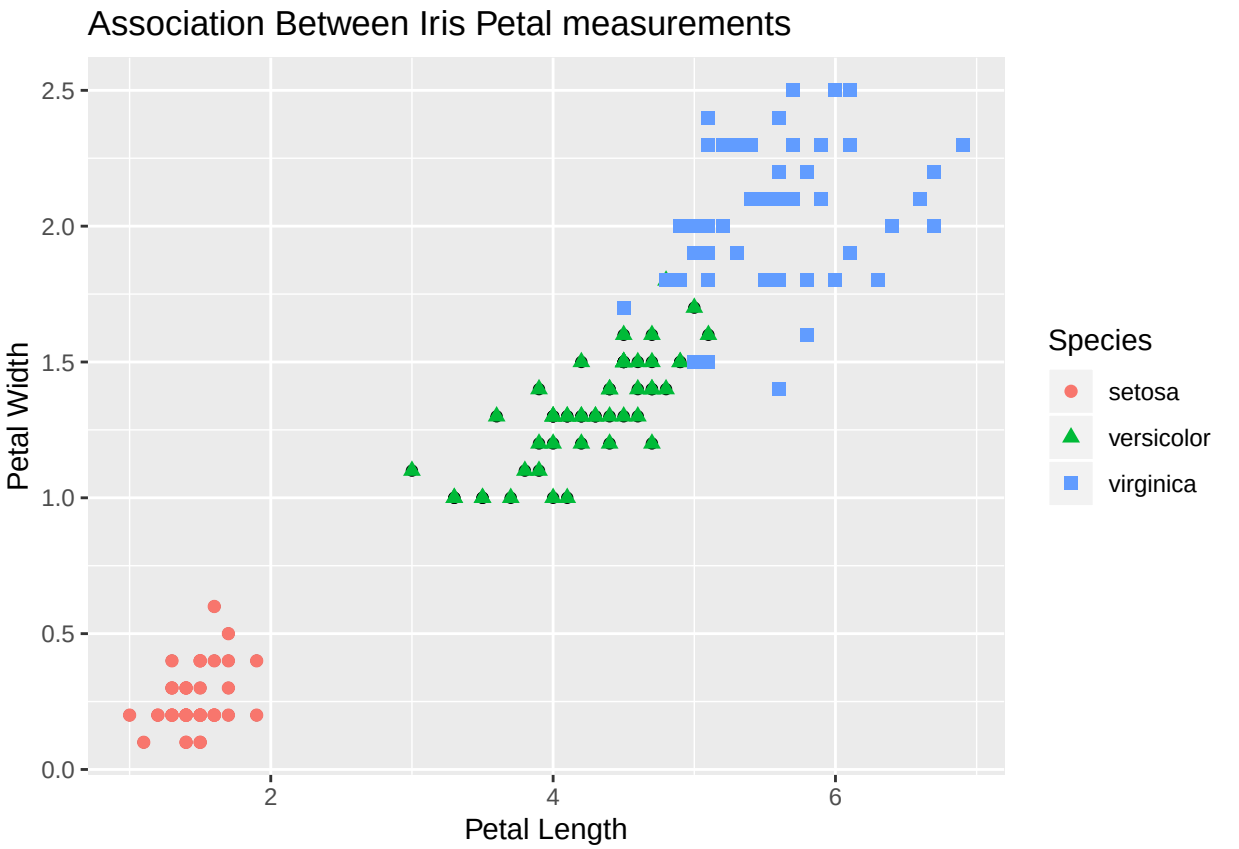
First a title and some axis labels. These are part of `labs()`.

```
#Getting fancy to show italics and greek symbols
iris_scatter <- iris_scatter +
  labs(title="Association Between Iris Petal measurements",
                    x="Petal Length", y="Petal Width")
iris_scatter
```



Now to add some colors, shapes etc to the point. Look at the `geom_point()` documentation for this.

```
iris_scatter <-  iris_scatter +
  geom_point(aes(color=Species, shape=Species),size=2)
iris_scatter
```

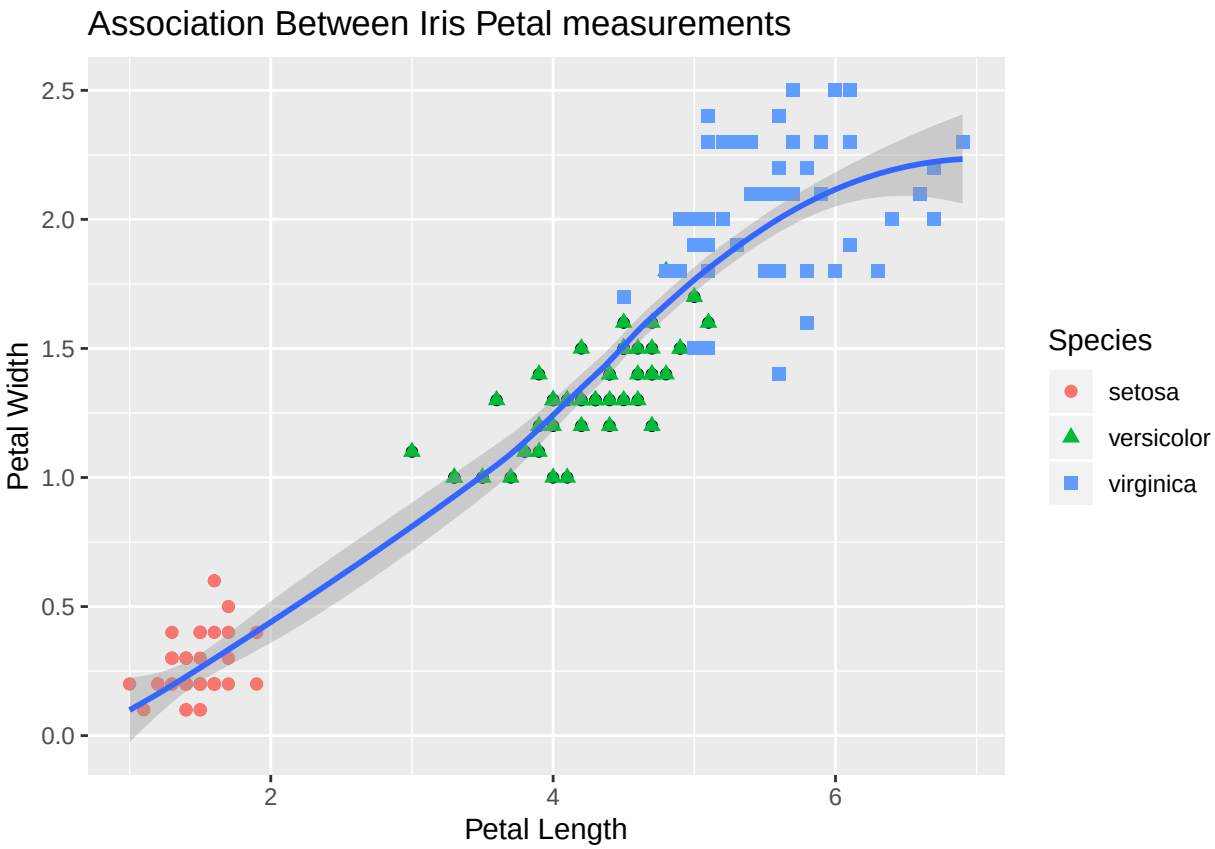


You'll notice we used `aes()` again, but this time inside of the geometry. This tells ggplot2 that this aes only applies to the points. Other geometries will not be affected by this.

In short, this is much easier than using base. Now `ggplot2` really shines when you want to add stats (regression lines, intervals, etc.).

Lets add a loess line with 95% confidence intervals

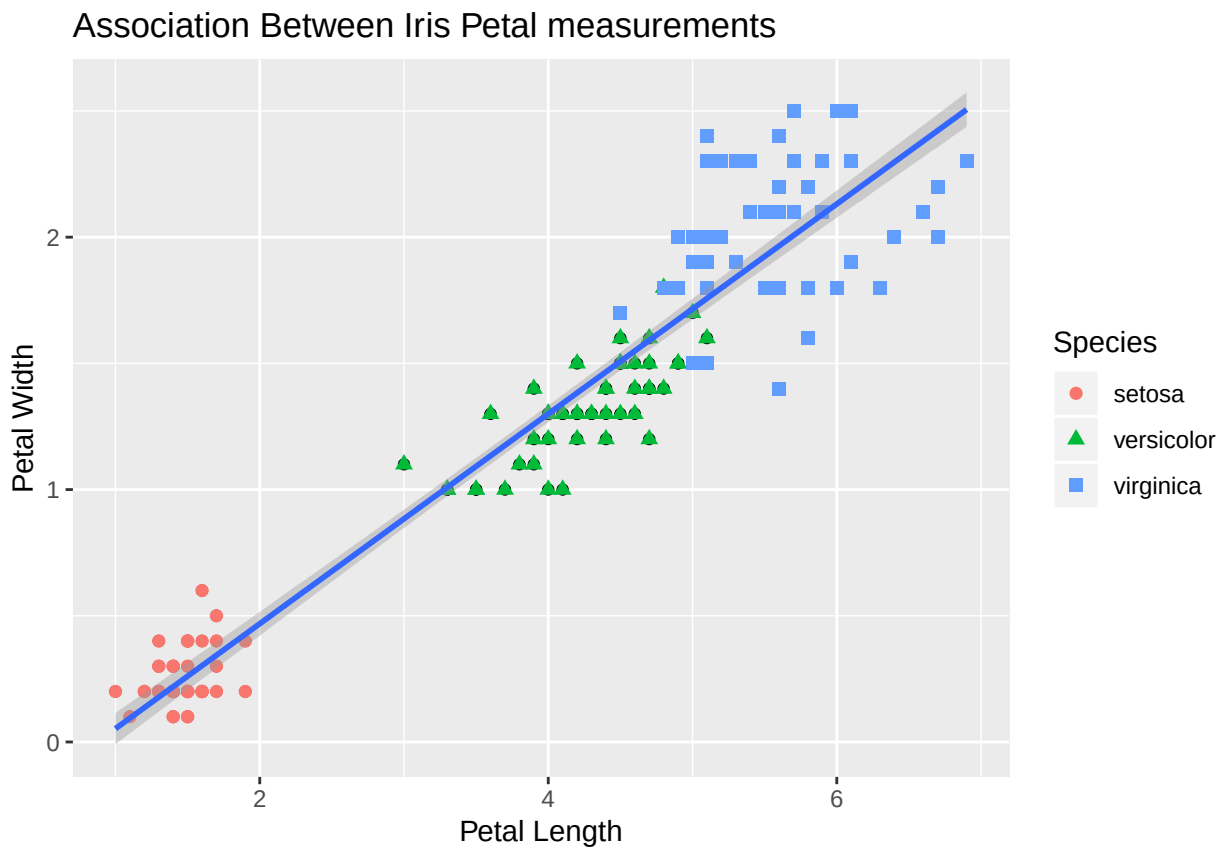
```
iris_scatter_loess <- iris_scatter +
  geom_smooth(method = "loess")
iris_scatter_loess
```



Try that in `base` with so little code!

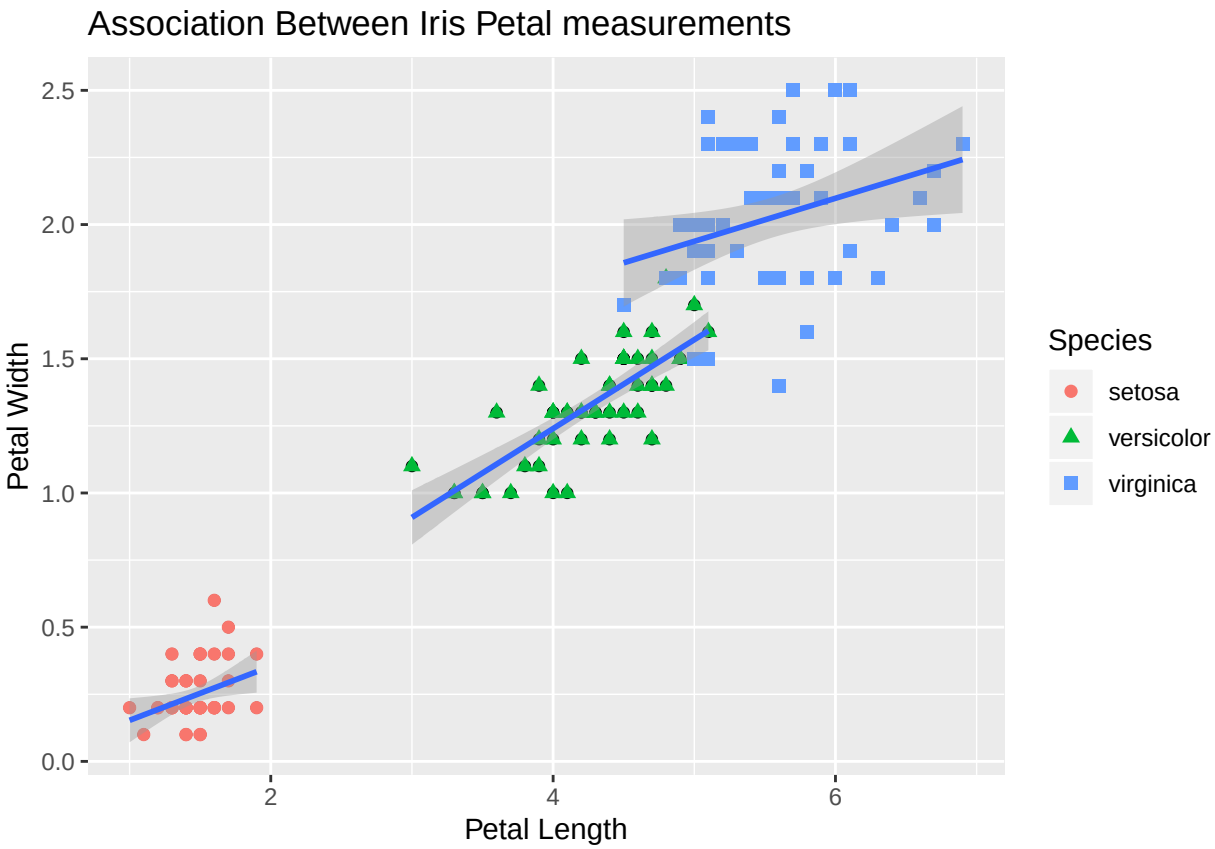
Or we could add a linear regression line with:

```
iris_scatter_lm <- iris_scatter +
  geom_smooth(method="lm")
iris_scatter_lm
```



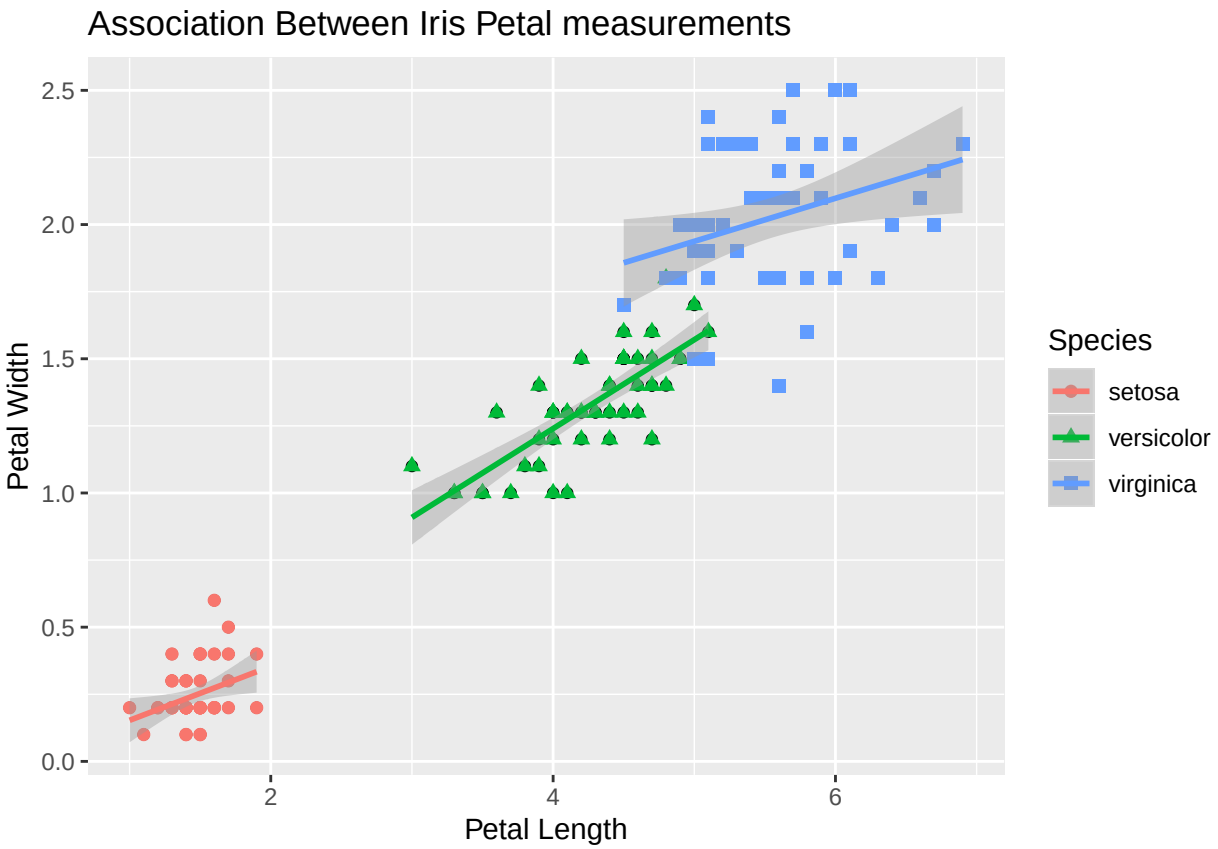
And if we are interested in the regressions by group we could do it this way.

```
iris_scatter_lm_group <- iris_scatter +
  geom_smooth(method="lm", aes(group=Species))
iris_scatter_lm_group
```



Or, if we wanted our regression lines to match the color.

```
iris_scatter_lm_color <- iris_scatter +
  geom_smooth(method="lm", aes(color=Species))
iris_scatter_lm_color
```



Notice, that we specified the `aes()` again, but for `geom_smooth()`. We only specified the x and y in the original `ggplot` object, so if want to do something different in the subsequent functions we need to overwrite it for the function in which we want a different mapping (i.e. groups).

In short, some of the initial setup for ggplot is a bit more verbose than base R, but when we want to do some more complex plots it is much easier in `ggplot2`.

Before we get into another exercise, lets look at some of the other geometries. The best place to do this is excellent `ggplot2` documentation of the geom functions.

Example explained

Now that we have the basics of `ggplot2` down, let's take a closer look at our example in `nla_analysis.R`.

Excercise 4.1

For this exercise we will work on creating a new plot from scratch. One of the concepts I hope to get across is that creating a plot is as much knowing data manipulation as it is knowing the details of your plotting system (`ggplot2` in our case). Add some new code at the end of our `nla_anlaysis.R` that does the following

1. Create a new data frame with the state by state average of total nitrogen, total phosphorus, and chlorophyll *a*
2. Using this newly created data frame, plot mean total nitrogen on the x-axis, mean total phosphorus on the y-axis, and size and color the points based on the chlorophyll (extra credit if you log transform these data)
3. Try to use `ggplotly` from the `plotly` package to create an interactive version of this plot.

4. Don't forget to comment your code

This will be a challenging excercise as it includes nearly all of the tidyverse components we have talked about.