Problem Set Debrief

Most of you did fantastically on the problem set. Mostly very well presented as well! Some common issues were:

- Presenting numbers in awkward units. Humans don't like to look at too many digits, so consider scaling/rounding.
- In general in tables, you should left align text, and right align numbers. These rules are not hard and fast.
- Titles on graphs.
- Don't repeat yourself (DRY). In programming, if you end up wanting to repeat code, you should almost always put that code into a function. It makes your work much more readable.
- In `knitr` chunks, you can use `message=FALSE` to suppress `R` messages.
- Words in your proofs will improve their presentation.
- Differing colors should only be used in figures if they convey extra information, like representing a third dimension in the data.
- Always put your final results in tables for presentation.
- You can use `\usepackage[margin=1in]{geometry}` in your preambles to get nicer margins.

The grammar of `ggplot2`

Now let's dig a little deeper into how `ggplot2` works.

`ggplot2` is an implementation of the "grammar of graphics," described in a book of the same title by Leland Wilkensen. The idea is that graphical representations of data, like language, have a logical grammatical structure. Most graphing packages ignore this structure and create one-off solutions for every different kind of graph that we might want to display. This is inefficient, and therefore displeasing to economists. `ggplot2` allows users to control the composition of statistical graphs by directly controlling the grammar of the graphical components.

Plots in `ggplot2` are built by putting together separate component parts. The three crucial components that we'll think about for now are:

1. data
2. aesthetics
3. layers/geometric shapes
4. themes

There are more, but these are the important ones. We'll tackle each separately.

Data

The *data* for `ggplot2` should always be packaged into a `data frame`. After loading the `ggplot2` library, we'll load the `R` iris dataset to demonstrate:

```
library(ggplot2)
data(iris)
## ggplot(data = iris, ... )
```

The first argument we pass to `ggplot()` will be the data frame that we intend represent graphically. This isn't the only way to get data into your `ggplot2` graphs, but is probably the best if you are graphing from a single `data.frame`.

Aesthetics

The second required argument for `ggplot()` is the aesthetic mapping/ of the plot. Aesthetics are used to map data to "things that you can see", such as the position of the data on the axes, the color, the shape, et cetera. Now we can create and display the `ggplot2` object `g` using our data an aesthetics.

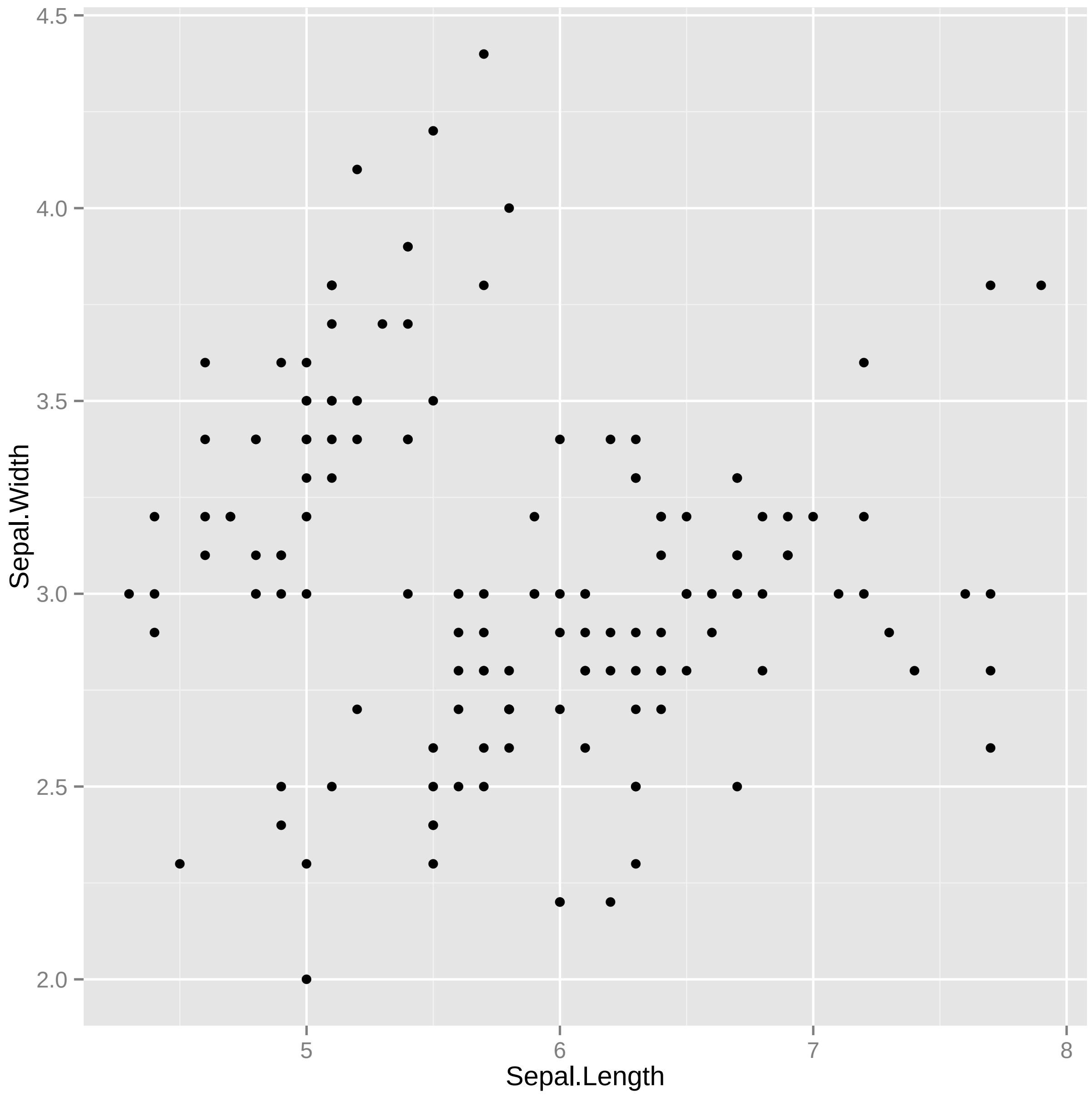
```
g <- ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width))
```

Or not. Why are we getting an error? Because we haven't specified any layers. So, in `ggplot2`, aesthetics are not the *only* thing that you see. Layers specify the class of graph that will be used to display the data.

Layers/geometric shapes

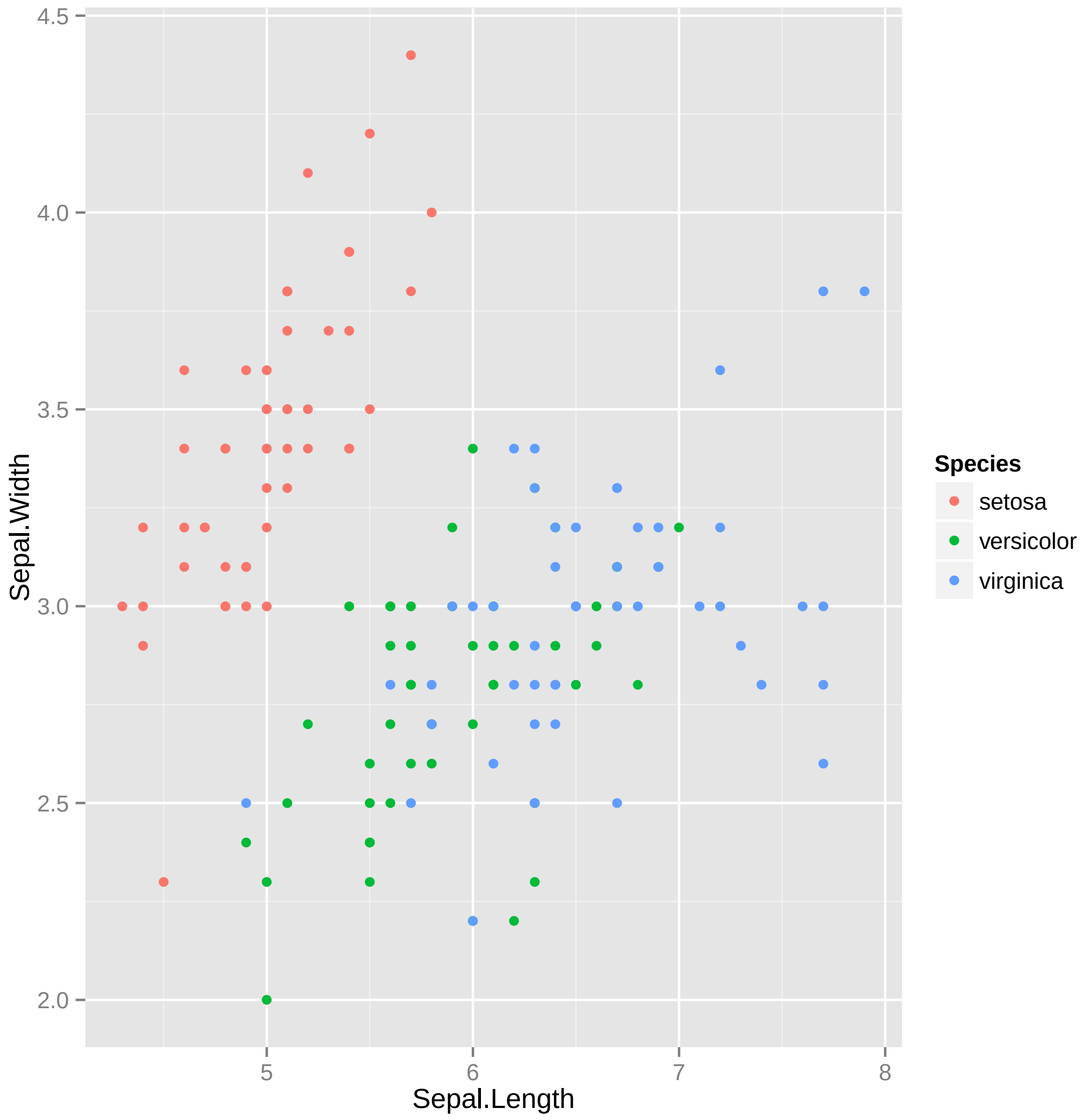
We've specified our data, and our data aesthetics, but not our *graphical layers* (i.e. geometric shapes, or `geoms`). Here, we'll add a layer of points:

```
g + geom_point()
```



We can also specify additional aesthetic options for each layer. Below, we'll tell `ggplot2` to graph the points again, this time specifying that each species should have a different color. Aesthetic options specified in the `ggplot()` function are the default for all layers, but aesthetics specified within layers can override the defaults for that layer only.

```r
g + geom_point(aes(color=Species))
```



Now, for a last plot, let's look at where `ggvis` is going to take us. For this, we'll revisit the kernel density of the OLS estimates from earlier.

```r
library(ggvis)
graphData[1:B,] %>% ggvis(x = ~beta) %>%
  layer_densities(
    adjust = input_slider(.1, 2, value = 1,
                          step = .1,
                          label = "Bandwidth adjustment"),
    kernel = input_select(
    c("Gaussian" = "gaussian",
      "Epanechnikov" = "epanechnikov",
      "Rectangular" = "rectangular",
      "Triangular" = "triangular",
      "Biweight" = "biweight",
      "Cosine" = "cosine",
      "Optcosine" = "optcosine"),
    label = "Kernel")
)
```

You will have to run this in RStudio for it to work (or a browser based document).

Initially, putting together the grammar of `ggplot2` may seem cumbersome. In fact, the code to construct simple scatterplots or histograms in `ggplot2` is almost certainly going to be more complex than a simple `plot()` or `hist()` from the base graphics package.[1]. But as your graphics needs become more complex, you will almost certainly find that `ggplot2` scales much better and is far more powerful than the base functions provided by `R`.

[1]In fact, `ggplot2` provides a function called `qplot()` that replicates the simpler syntax from the base graphing package, if you prefer.