

Group Project 1

Biology 368/664 Bucknell University

Ian Mawn, Kaitlin McLain, and Emily Scholfield

21 Feb 2022

R for Dummies (BIOL208 Edition)

Welcome to this R tutorial for BIOL208! This is designed for someone who is brand new to R, so we are going to start with the very basics. To make this interesting, we'll be looking at a dataset on irises since plants are extremely important to the study of ecology and evolution. To best follow along, you should have a new markdown file open in R studio where you can copy and paste line of code from this tutorial for you to practice running on your own.

Typically, when doing data analysis in R, we want to begin by loading in the data and packages that we will use.

Loading Libraries and Data

R has many built in functions that will allow us to perform all kinds of actions. However, sometimes it is useful to load some additional functions that other R users have created. We do this by loading in libraries. In the code chunk below (the grey box; this is where code is typed and run) you can see that I have loaded in some libraries.

```
# Install packages and load their libraries here.
if (!require("tidyverse")) install.packages("tidyverse"); library(tidyverse)
if (!require("cowplot")) install.packages("cowplot"); library(cowplot)
if (!require("UsingR")) install.packages("UsingR"); library(UsingR)
if (!require("ggpubr")) install.packages("ggpubr"); library(ggpubr)
if (!require("readxl")) install.packages("readxl"); library(readxl)
```

Libraries: For some examples, one library is called tidyverse. It contains a ton of useful tools, but the one most relevant to this tutorial is called **ggplot**. **ggplot** allows us to create all kinds of plots to visualize our data, which you will learn more about below. You can also see the library called **cowplot**. **cowplot** is useful as it allows us to easily change the aesthetics of our graphs, and keep the designs consistent between different graphs. Finally, you can see the library **readxl**. This is useful for exactly what it sounds like... reading Microsoft Excel files! This brings us to the next step, entering data into R.

Data: In the code chunk below I am going to read in some data regarding features of different species of iris (the flower). There are a few main parts of the code to think about when loading in data. First, the **read_csv()** function tells R to load in a comma separated values (csv) file with the name of the file in the parentheses. Here, you can see that our data is called "iris.csv". It is very important to type the file name exactly as it is saved on your computer, otherwise R will not be able to find it. Once the data is loaded in, we need to call it something. That is what the "IrisData <-" part of the code means. In English, it translates to: IrisData is the data found in the csv file "iris.csv". When you read in a data file, you will see it appear in your "Environment" on the right of R studio. If you click on "IrisData" in your Environment, a new tab will appear with all of the data.

```

IrisData <- read_csv ("iris.csv")

## Rows: 150 Columns: 5

## -- Column specification -----
## Delimiter: ","
## chr (1): variety
## dbl (4): sepal.length, sepal.width, petal.length, petal.width

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

What happens if your data is in an excel file instead of a csv file? Fear not! Data can also be loaded in from an excel file using the `read_excel()` function instead of `read_csv()`. In a normal data analysis, you would only read in the data once using *either* the `read_csv()` *or* `read_excel()` functions. Your data will only read in properly if the file type matches the function you're using, so pay attention to how your data is saved before you read it in!

Using R (The Basics)

Now that we have the pieces we need, we can start learning about R. You have already seen a few aspects of R in the last section, but here we will go into some more detail about the basic functions of R to get you started with analyzing data.

This document is an R markdown file. The highlights of a markdown file is that you can have mostly regular text (not code) but still include code and their output in specific sections. This is especially useful as it allows for lots of explanation of what the code does, like we need to do here for a tutorial. Markdowns are also convenient because they can be converted to many different formats such as a PDF (like you're reading), an html, and others.

As mentioned before, the grey sections are called code chunks. This is where the actual code is written and ran. You can insert a code chunk to your markdown file by clicking the green box with "+C" at the top of R studio. When you do this, a drop-down menu appears where you can choose a coding language (since this is about R, we will only use that). Refer to the image below if you can't find the code chunk button.

In the top line of your new code chunk, you'll see "{r}". This can be used to name your code chunks if you wish. This can help keep your thoughts organized and is actually an important part of knitting your file... Say what??? We're going to be knitting? Don't worry, we'll explain *all* the details about knitting in the later sections of this tutorial. For now just know that naming your code chunks can help organize your thoughts and can save you problems with knitting later down the road if you make sure to name your each of your code chunks something unique.

```

#Example code chunk
4+1

```

```
## [1] 5
```

```
#This line will not run
```

Code within chunks can be run in a few ways:

1. In the top right of a chunk is a green play button that will run every line of code in the chunk.
2. Placing your cursor on a line you want to run, or highlight it, and hit command + enter (or control + enter).
3. Click the run button in the top of the screen, which will bring up a few options for running code. You can run all of the code chunks in your document at once (in order from top to bottom) with this option. This is useful for ensuring your code is functional when finalizing a document.

Looking at the example code chunk above, there are a few points to make about coding in R. First, you can see a green line that starts with a "#". The hashtag/number/pound symbol tells R that this line is a comment, and

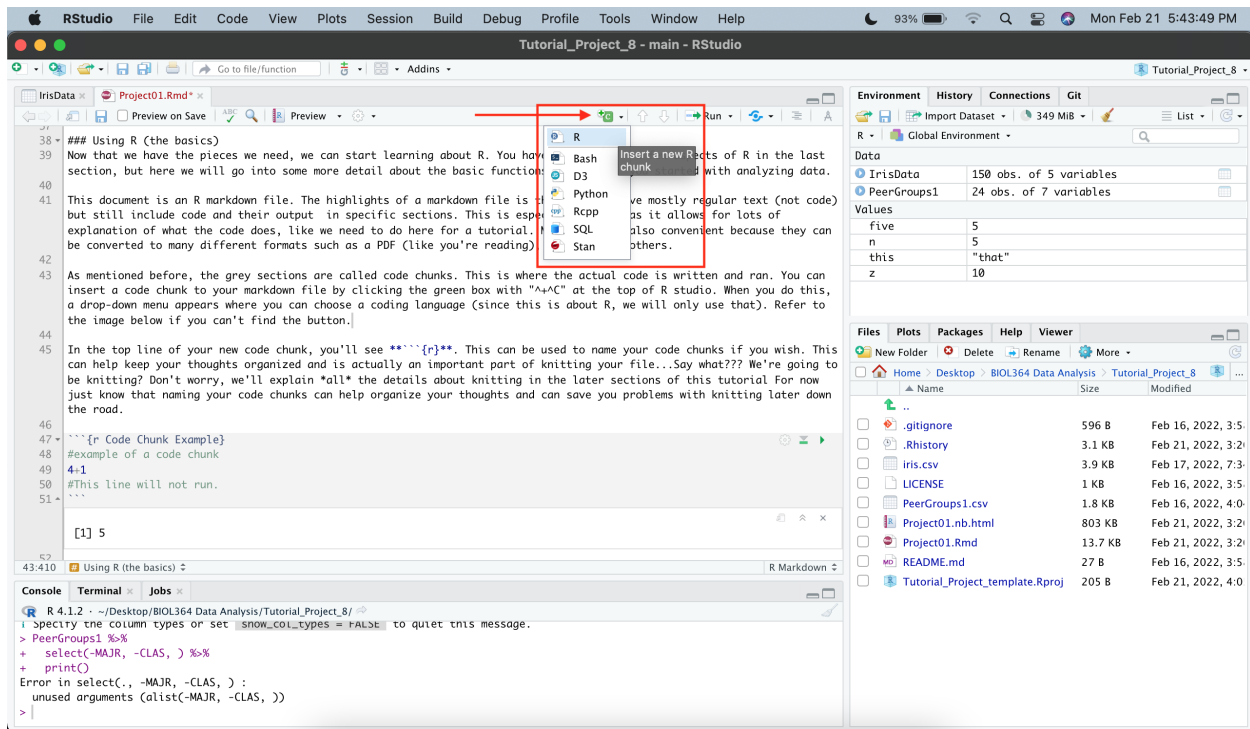


Figure 1: Adding a New Code Chunk

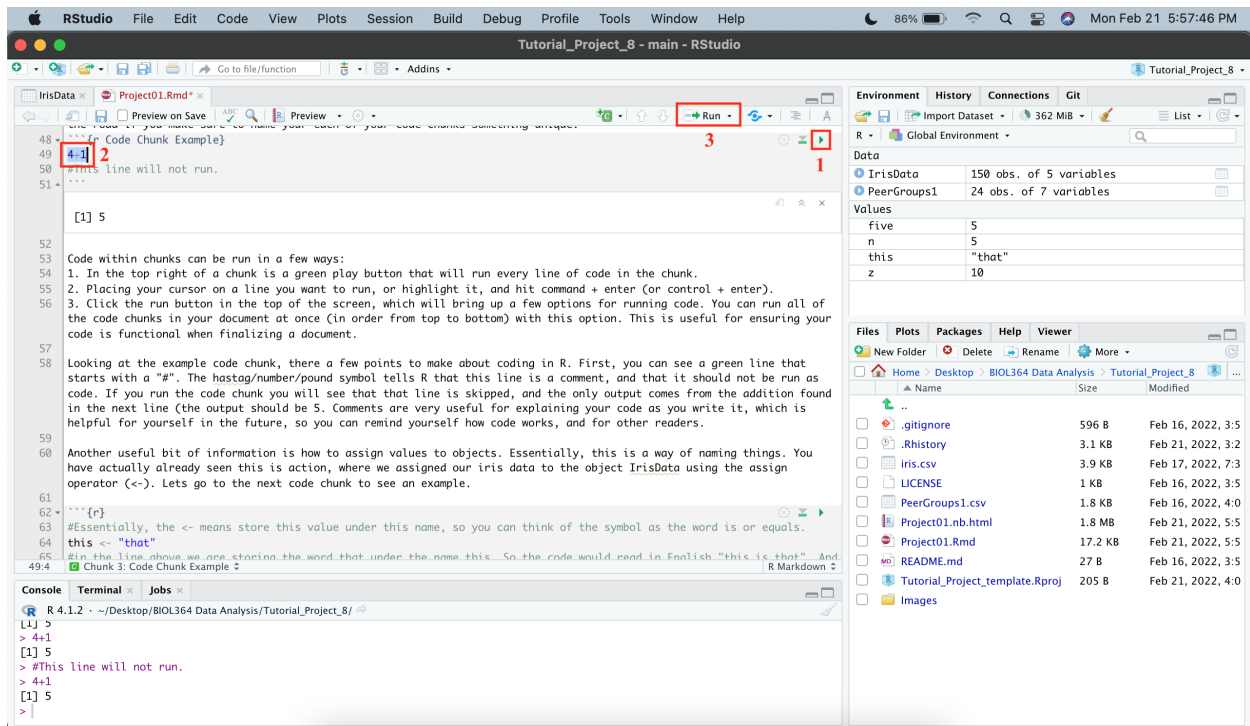


Figure 2: Running Code

that it should not be run as code. If you run the code chunk you will see that that line is skipped, and the only output comes from the addition found in the next line (the output should be 5 since $4+1=5$). Comments are very useful for explaining your code as you write it. This is helpful for yourself in the future, so you can remind yourself how different code works, and for other people reading through or referencing your code.

Another useful bit of information is how to assign values to objects. Essentially, this is a way of naming things. You have actually already seen this in action, where we assigned our iris data to the object IrisData using the assign operator `<-`. Lets go to the next code chunk to see an example and notice how all of these values/variables will get stored in the Environment.

```
#Essentially, the <- means "store this value under this name," so you can think of the symbol as the word
this <- "that"
```

```
#In the line above we are storing the word that under the name this. So the code would read in English
this
```

```
## [1] "that"
```

```
#Other examples
```

```
n <- 5
z <- 10
n+z
```

```
## [1] 15
```

```
z/n
```

```
## [1] 2
```

```
(z+n)/3 -> five
five * 10
```

```
## [1] 50
```

This is especially useful for storing data under a specific name, like we did at the beginning with the iris data. If you run the line of code below, it will print out all of the iris data.

```
IrisData
```

```
## # A tibble: 150 x 5
##   sepal.length sepal.width petal.length petal.width variety
##         <dbl>      <dbl>      <dbl>      <dbl> <chr>
## 1         5.1         3.5         1.4         0.2 Setosa
## 2         4.9         3         1.4         0.2 Setosa
## 3         4.7         3.2         1.3         0.2 Setosa
## 4         4.6         3.1         1.5         0.2 Setosa
## 5         5         3.6         1.4         0.2 Setosa
## 6         5.4         3.9         1.7         0.4 Setosa
## 7         4.6         3.4         1.4         0.3 Setosa
## 8         5         3.4         1.5         0.2 Setosa
## 9         4.4         2.9         1.4         0.2 Setosa
## 10        4.9         3.1         1.5         0.1 Setosa
## # ... with 140 more rows
```

We can also access specific variables within the data using the `$` symbol. So if we only wanted to see the data about petal length, we can run the following code:

```
IrisData$petal.length
```

```
## [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
## [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
```

```
## [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
## [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
## [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
## [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
## [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
## [127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
## [145] 5.7 5.2 5.0 5.2 5.4 5.1
```

Those are the basic functions you should know about R moving forward. Now we can explore more tools you can use to explore and analyze a set of data.

Exploring Your Data

Once we have our data, it is important to explore it before beginning analysis. First, you want to come up with the question that you will attempt to answer *before analyzing your data*. This is important since we want to remain objective throughout the data analysis process.

For this tutorial, we're interested in the following questions:

1. Are sepal and petal dimensions correlated within a single species?
2. Do sepal and petal dimensions differ between Iris species?

One way to begin to understand your dataset is to run the **str()** (structure) function. This outputs lots of information about the dataset. It will show you each column of the data, the class of the column (what type of data it is: a number (num), a word (or character: chr), etc), the number of observations in the column, and some of the data within each column. This is useful for ensuring that each column contains the data you expect. For example, one column in the dataset is sepal length, so I would expect to find in the output of **str()** that there is a column called sepal length (it is actually called `sepal.length`, but that is good), it should be a number, there should be 150 observations, and the data should actually be numbers.

In the chunk below I will run **str()** on the iris dataset to confirm my expectations and make sure that the data looks right.

```
str(IrisData)

## spec_tbl_df [150 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ sepal.length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ sepal.width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ petal.length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ petal.width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ variety      : chr [1:150] "Setosa" "Setosa" "Setosa" "Setosa" ...
## - attr(*, "spec")=
## .. cols(
## ..   sepal.length = col_double(),
## ..   sepal.width = col_double(),
## ..   petal.length = col_double(),
## ..   petal.width = col_double(),
## ..   variety = col_character()
## .. )
## - attr(*, "problems")=<externalptr>
```

One thing that stands out to me after running **str()** is the “variety” column. For starters, you may want to rename the column to something that’s easier for you to understand, such as “species.” You can do this using the **rename()** function as explained below. Once you’ve run that code, you can click on the dataset in your Environment and should see the column has been renamed.

```
#The rename() function allows this format: DataName <- rename(DataName, NewVariableName = OldVariableName)
IrisData <- rename(IrisData, species = variety)
```

If we want to compare variables between species of irises, this column will be very useful. To make those comparisons easier, I am going to tell R that this variable is categorical. We can group the data into those categories using the `as.factor()` function below.

```
IrisData$species <- as.factor(IrisData$species)
#This code tells R to replace the current character species column of IrisData with the new factor species

str(IrisData)
```

```
## spec_tbl_df [150 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ sepal.length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ sepal.width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ petal.length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ petal.width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ species      : Factor w/ 3 levels "Setosa","Versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
## - attr(*, "spec")=
## .. cols(
## ..   sepal.length = col_double(),
## ..   sepal.width = col_double(),
## ..   petal.length = col_double(),
## ..   petal.width = col_double(),
## ..   variety = col_character()
## .. )
## - attr(*, "problems")=<externalptr>
```

#Running str() again shows that species is now a factor with 3 levels which correspond to the 3 different species

It can also be useful to look at the top and bottom of your dataset using the `head()` and `tail()` functions, which show the first 6 rows and last 6 rows respectively. This is useful to ensure that the data makes sense (the data is numeric if you were expecting that) and to make sure that the correct number of rows are present (for example, make sure that the last few rows are not empty). Or, if the data is time dependent, you can use these functions to make sure that the first times are in the first row and the last times are at the end.

#This chunk will output two functions, and it will separate them into two windows if you run the whole chunk

```
## # A tibble: 6 x 5
##   sepal.length sepal.width petal.length petal.width species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 Setosa
## 2         4.9         3         1.4         0.2 Setosa
## 3         4.7         3.2         1.3         0.2 Setosa
## 4         4.6         3.1         1.5         0.2 Setosa
## 5         5         3.6         1.4         0.2 Setosa
## 6         5.4         3.9         1.7         0.4 Setosa
```

```
tail(IrisData)
```

```
## # A tibble: 6 x 5
##   sepal.length sepal.width petal.length petal.width species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         6.7         3.3         5.7         2.5 Virginica
## 2         6.7         3         5.2         2.3 Virginica
## 3         6.3         2.5         5         1.9 Virginica
## 4         6.5         3         5.2         2 Virginica
## 5         6.2         3.4         5.4         2.3 Virginica
## 6         5.9         3         5.1         1.8 Virginica
```

A very useful function for data exploration is `summary()`. This outputs the mean, median, min, max, and first and third quartiles of any numeric data, and the number of observations within our variables. This is an important step in any data exploration. This can be useful for validating your data against external sources. If external sources show that the average sepal length of irises is 100 cm long, but our sepal lengths range from 4.3 to 7.9 cm, something might be wrong. Luckily the average is closer to 5-7 cm, so our data does make sense.

```
summary(IrisData)
```

```
##      sepal.length      sepal.width      petal.length      petal.width
## Min.      :4.300    Min.      :2.000    Min.      :1.000    Min.      :0.100
## 1st Qu.:5.100    1st Qu.:2.800    1st Qu.:1.600    1st Qu.:0.300
## Median :5.800    Median :3.000    Median :4.350    Median :1.300
## Mean   :5.843    Mean   :3.057    Mean   :3.758    Mean   :1.199
## 3rd Qu.:6.400    3rd Qu.:3.300    3rd Qu.:5.100    3rd Qu.:1.800
## Max.   :7.900    Max.   :4.400    Max.   :6.900    Max.   :2.500
##           species
## Setosa      :50
## Versicolor:50
## Virginica   :50
##
##
##
```

Visualization with Graphs

Now that we've spent a substantial amount of time exploring and organizing our data, we're ready to start analyzing it! The first step here is to visualize our data since up until this point, all we've been looking at are numbers (which can be hard for identifying relationships and patterns between variables). When considering what graphs would be best for visualizing our data, we should consider our questions of interest and what types of variables we're dealing with. As a reminder, our two questions of interest are below:

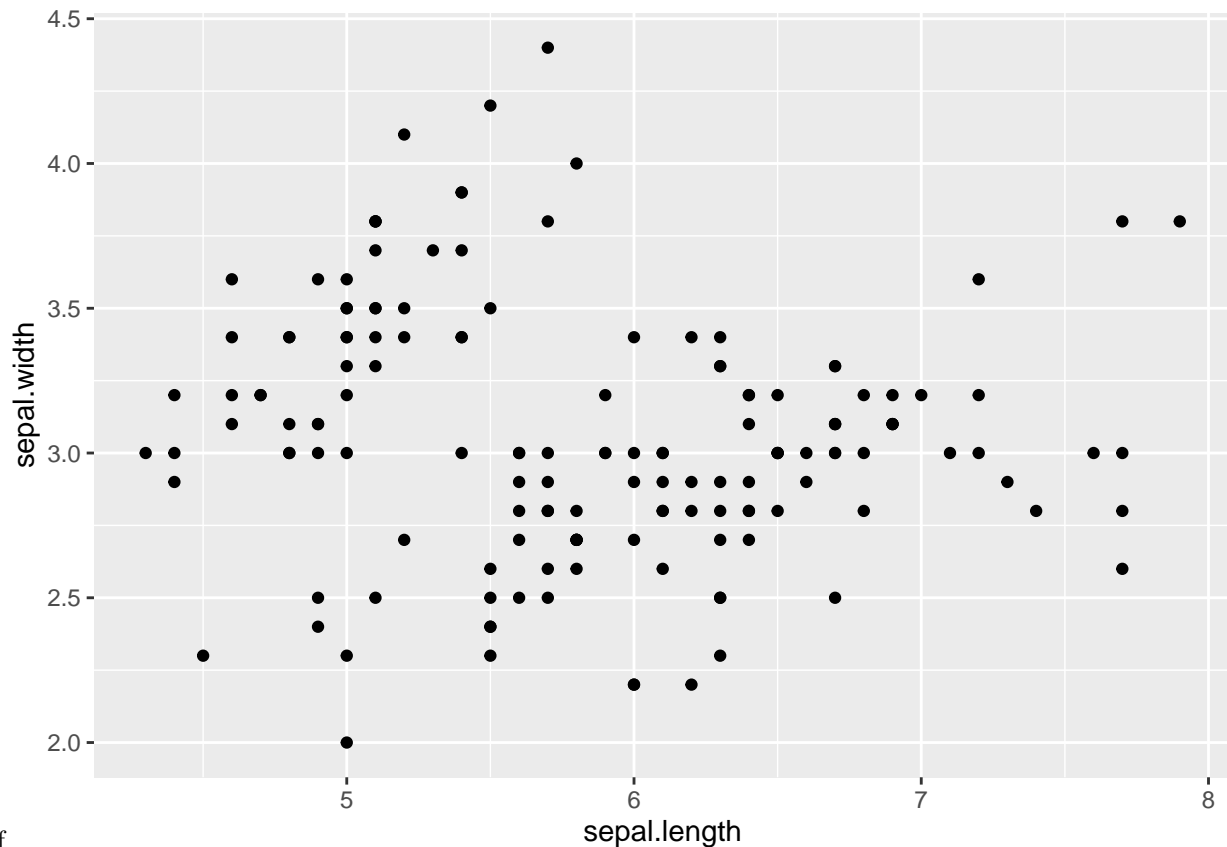
1. Are sepal and petal dimensions correlated within a single species?
2. Do sepal and petal dimensions differ between Iris species?

Are sepal and petal dimensions correlated within a single species? Let's focus on question number one first. There are a couple different, more specific questions we can look at within this. Such as, the relationship between sepal length and sepal width, or petal length and petal width. Perhaps we can even compare sepal length and petal length, or sepal width and petal width. Wow! There are a ton of different relationships to look at. For our purposes, let's narrow in on two relationships:

1. Sepal length and sepal width
2. Sepal length and petal length

Since all of these variables are numerical, a scatterplot would best represent the data. The `ggplot` function is extremely useful for creating graphs. Below is the code for creating a basic scatterplot.

```
#Follows the form: ggplot(DataName, aes(x=VariableOne, y=VariableTwo)) + geom_point()
ggplot(IrisData, aes(x=sepal.length, y=sepal.width)) + geom_point()
```

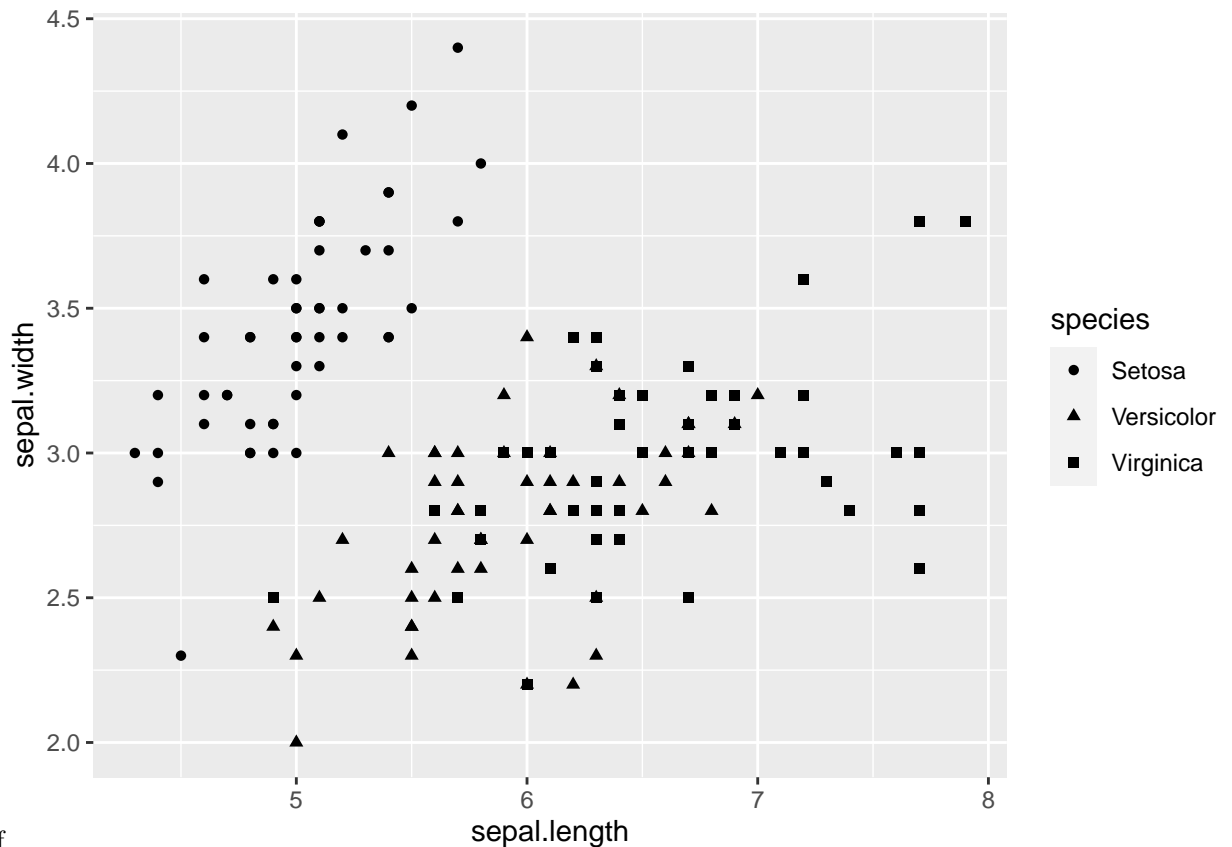


Scatterplot-1.pdf

#Learning check: Try running the code without + geom_point(). What happens?

In the output we see a basic scatterplot without any clear pattern. This could be because we're looking at data for all three species. Remember, our original question was about whether there were relationships between the variables within a *single* species. We can either filter the data so that we can plot one species at a time, or we can just color-code the points on the scatterplot by species so we can still look at all of the data (this is the method we'll use).

```
ggplot(IrisData, aes(x=sepal.length, y=sepal.width, shape=species)) + geom_point()
```

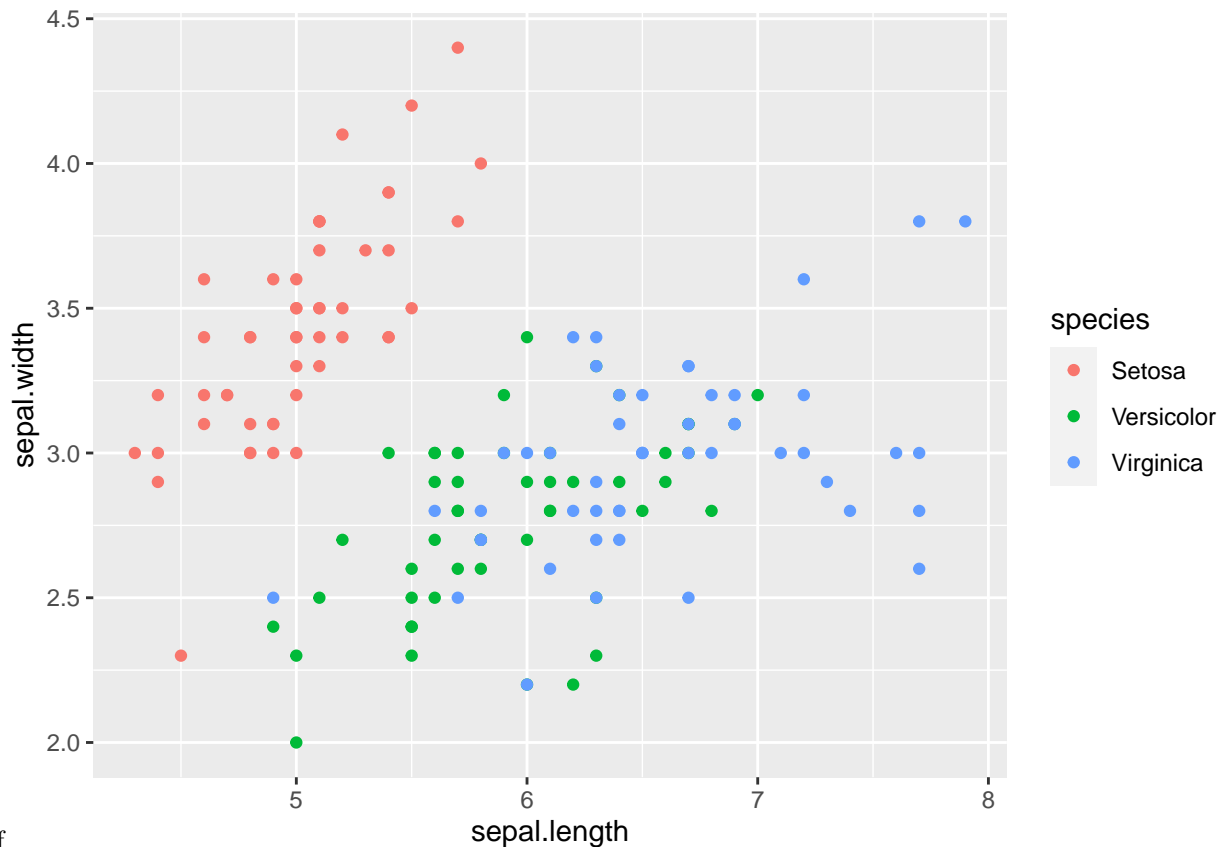



by Species-1.pdf

#Adding shape=species in the code line tells R to group the points according to species and to differen

By separating the data points according to species we can start to see some patterns emerging between the variables within each species. However, it still takes a second to differentiate between Versicolor and Virginica since they occupy the same general space on the plot and you don't immediately notice the difference in shapes. To make it easier to distinguish between the two, we can use color to differentiate the species rather than shape.

```
ggplot(IrisData, aes(x=sepal.length, y=sepal.width, color=species)) + geom_point()
```

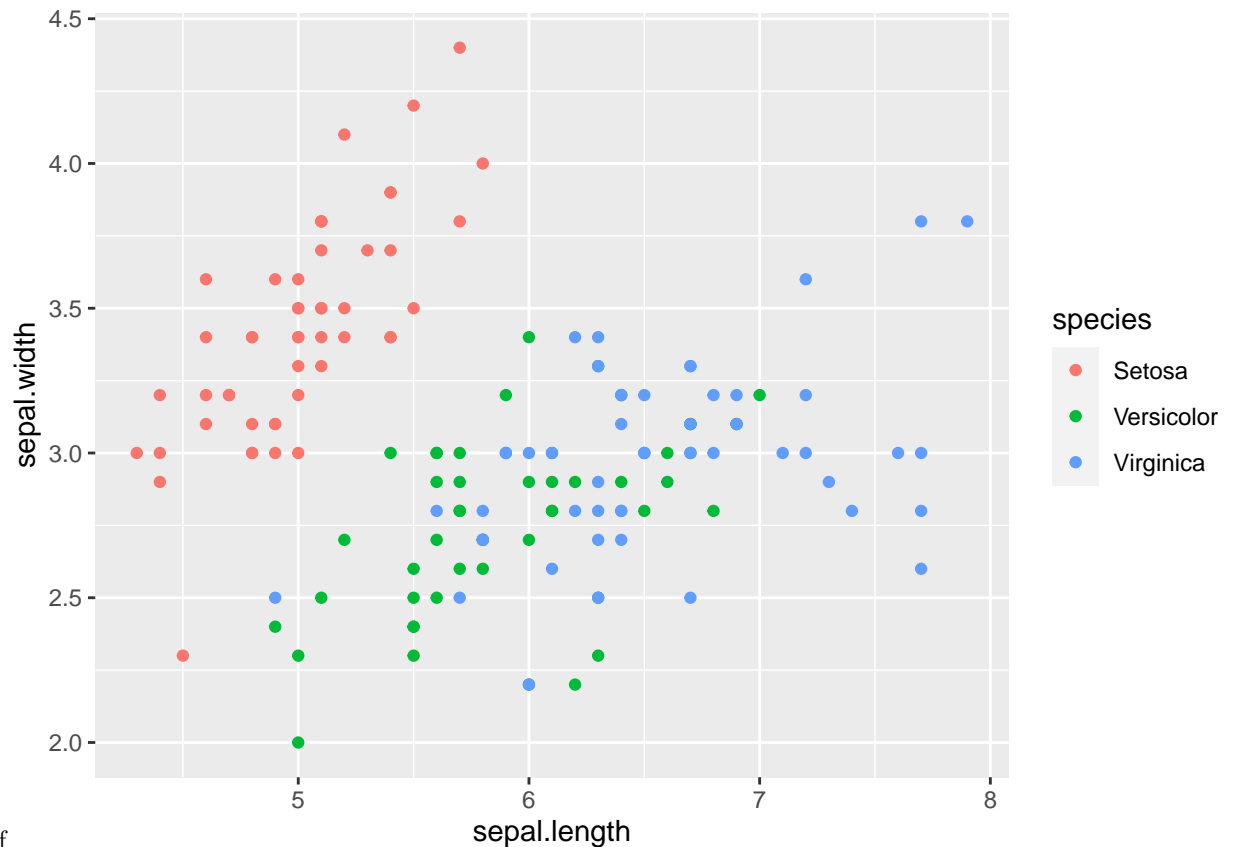


by Species-1.pdf

#Adding color=species in the code line tells R to differentiate the species by color as well as by shape

Based on this scatterplot, there seems to be a positive correlation between sepal length and sepal width within each species. Say you wanted to present this data in a formal report. You would likely want to clean it up a bit. Fortunately, **ggplot** has a *ton* of ways you can spruce up your graphs to look neater and more professional. Let's take a look at some of those tools.

```
ggplot(IrisData, aes(x=sepal.length, y=sepal.width, color=species)) +  
  geom_point()
```



it up-1.pdf

#This code runs the same as the code we used previously, even though geom_point() is on a separate line

```
ggplot(IrisData, aes(x=sepal.length, y=sepal.width, color=species)) + #Basic scatterplot grouped by color
  geom_point() + #Adds data points to plot
  geom_smooth(method=lm, se=FALSE) + #Adds regression lines
  labs(x = "Sepal Length (cm)", y = "Sepal Width (cm)") + #Renames axis labels
  labs(col="Species") + #Renames legend
  labs(title = "Sepal Width vs. Sepal Length in Three Iris Species") + #Gives the plot a title
  theme(plot.title = element_text(hjust = 0.5)) #Centers the plot title
```

`geom_smooth()` using formula 'y ~ x'



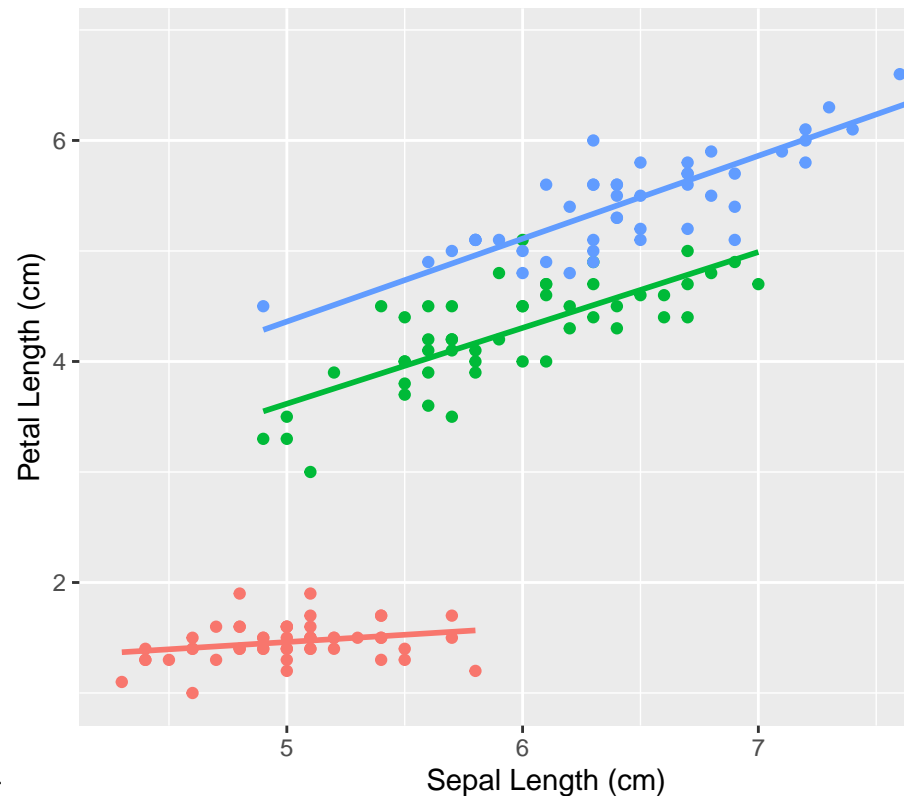
it up-2.pdf

This plot looks much nicer and is easier to interpret compared to our initial scatterplot. Now that we have the code all figured out, we can essentially copy and paste the same code but substitute in new variable to look at other relationships as well. Let's look at the relationship between

```
ggplot(IrisData, aes(x=sepal.length, y=petal.length, color=species)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE) +
  labs(x = "Sepal Length (cm)", y = "Petal Length (cm)") +
  labs(col="Species") +
  labs(title = "Petal Length vs. Sepal Length in Three Iris Species") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```

Petal Length vs. Sepal Length in Three Iris Species

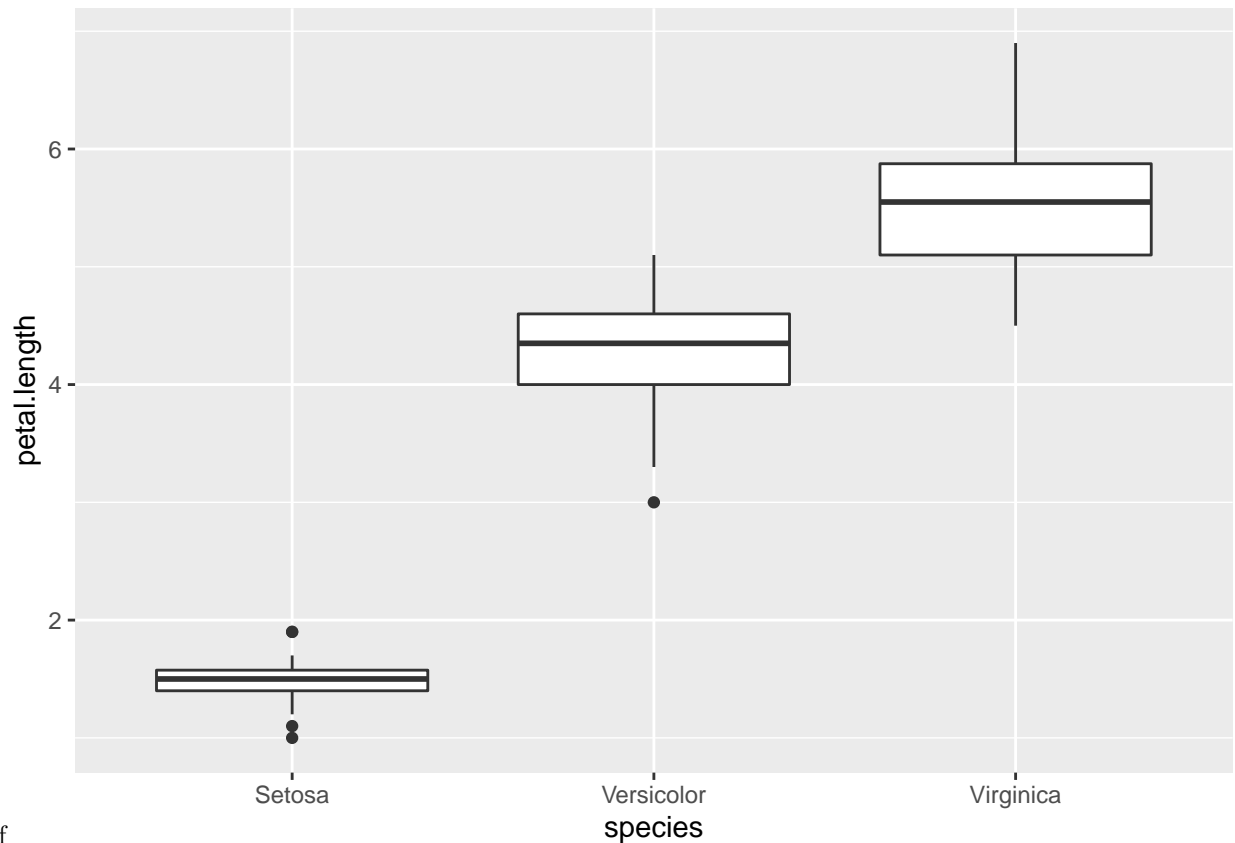


comparing Petal Length and Sepal Length-1.pdf

This scatterplot reveals that there also seems to be a positive correlation between sepal length and petal length within each species, though the strength of that relationship may differ between species. Feel free to use this scatterplot code to visualize relationships between other variables as well.

Do sepal and petal dimensions differ between Iris species? Our next question of interest deals with comparing dimensions *between* species. Once again, there are a ton of different, more specific questions we can look at within this. For our purposes, let's focus on sepal length between species. Since species is a categorical variable, a boxplot would best represent the data. The code for a basic boxplot is below.

```
ggplot(IrisData, aes(x=species, y=petal.length)) + geom_boxplot()
```



boxplot-1.pdf

#Replacing geom_point() from our basic scatterplot code with geom_boxplot() tells R to make a boxplot.

There appears to be a difference in sepal length between species (note: determining whether or not that difference is significant is for later statistical testing). Just as we did before, let's spruce up our plot a bit.

```
ggplot(IrisData, aes(x=species, y=petal.length, fill=species)) + #Addition of fill=species adds color
  geom_boxplot() + #Specifies boxplot
  labs(x = "Species", y = "Petal Length (cm)") + #Renames axes
  labs(title = "Petal Length for Three Iris Species") + #Gives plot a title
  theme(plot.title = element_text(hjust = 0.5)) + #Centers title
  theme(legend.position = "none") #Removes legend
```



Boxplot-1.pdf

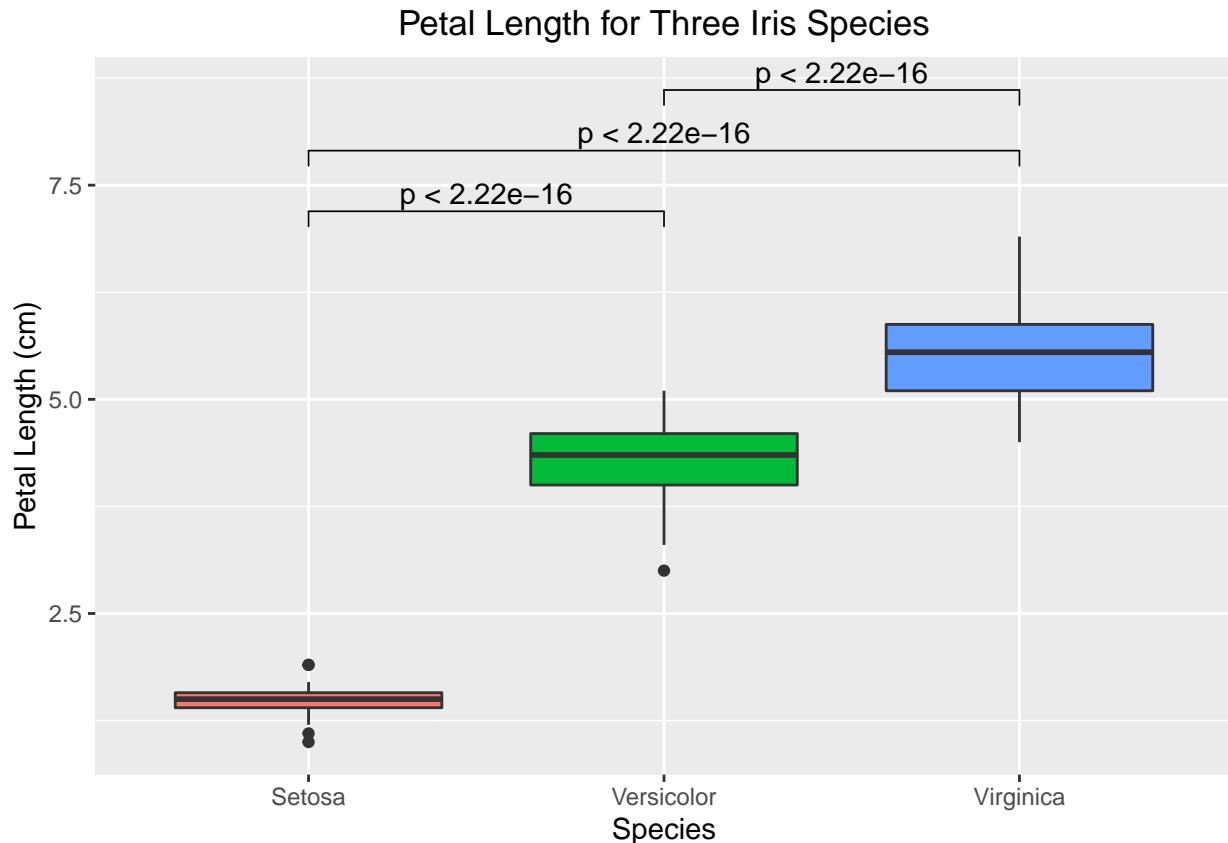
Once you get *really* good with R, you'll be able to make plots like the one below, where you include the significance bars for pairwise comparisons between each of the species. Even though we're getting ahead of ourselves here, feel free to play around with the code below to make plots for other variables and look at the p-values for pairwise comparisons between species.

```
compare_means(petal.length ~ species, data = IrisData)
```

```
## # A tibble: 3 x 8
##   .y.      group1    group2      p    p.adj p.format p.signif method
##   <chr>    <chr>    <chr>    <dbl>  <dbl> <chr>    <chr>    <chr>
## 1 petal.length Setosa    Versicolor 5.65e-18 1.7e-17 <2e-16  ****    Wilcoxon
## 2 petal.length Setosa    Virginica  5.67e-18 1.7e-17 <2e-16  ****    Wilcoxon
## 3 petal.length Versicolor Virginica  9.13e-17 9.1e-17 <2e-16  ****    Wilcoxon
```

```
my_comparisons <- list(c("Setosa", "Versicolor"), c("Setosa", "Virginica"), c("Versicolor", "Virginica"))
```

```
ggplot(IrisData, aes(x=species, y=petal.length, fill=species)) + #Addition of fill=species adds color
  geom_boxplot() +
  labs(x = "Species", y = "Petal Length (cm)") + #Renames axes
  labs(title = "Petal Length for Three Iris Species") + #Gives plot a title
  theme(plot.title = element_text(hjust = 0.5)) + #Centers title
  theme(legend.position = "none") + #Removes legend
  stat_compare_means(comparisons = my_comparisons)
```



Hypothesis Testing with Statistical Tests

Now we can move on to analyzing our data more critically with statistical testing for specific hypotheses. When using statistical tests, always be sure to check the assumptions made by the particular test you plan to use. This is important because many common statistical tests, including the two-sample t-test we will use here, assume that the datasets being compared are normally distributed. While much biological data is normally distributed, this is not always true, so be sure to explore and understand the behavior of your data before performing tests.

```
#shapiro.test()
```

If the data you are using is not normally distributed, test to see if a logarithmic transformation would make the distribution normal. If so, you should use the log-transformed data. If not, continue on with the original, non-transformed data but note in your report that any statistical test results should be interpreted with caution, given that the base requirements for accurate use of the test have not been met.

```
#LogPetalLength <- log(petal.length)
```

However, each test which assumes normality also has a sample size requirement which can allow for them to produce accurate results using non-normal data. For a two-sample t-test, each group is required to have a sample size of at least 15 observations to overcome the need for normality. This cutoff is different for each type of test. Thus, if your samples fit this requirement, you can proceed without worrying about normality.

Additional Tools

```
####Explanation of p-values
```

A p-value represents the probability that the null hypothesis is true given the outcome observed. For most purposes, a p-value of 0.05 or lower is considered statistically significant. However, staunchly relying on this value comes with some concerns, because although it is commonly used and accepted, it is essentially

an arbitrary cutoff. Thus, be sure to understand what this cutoff means so you can accurately express the strength of each result.

Remember, a p-value of 0.05 means that there is a 5% chance of the observations being as they are if the null hypothesis is true. Thus, if you test 20 variables, it is statistically likely that at least one of them will show significance simply by chance.

R Markdown Formatting Tricks (& Knitting)

GitHub

Grading

Each student will be expected to complete the following tasks to earn 85% of the points available for this assignment (21/25).

- Identify and obtain suitable dataset
- Use a Github repository and version control to collaborate on the project
- Spend 4-6 hours preparing, coding, and testing tutorial
 - Data exploration
 - Data visualization
 - Hypothesis testing
- Present tutorial in class
- Provide public archive suitable for sharing to students/faculty

Tutorials from previous classes can be viewed at our public github site: <https://github.com/Bucknell-Biol364>

Each group should use an *Acknowledgements* section to document the participation of each member and the collaboration within and between groups.

Additional credit will be awarded for providing assistance to other groups and for the development of a tutorial that goes beyond the minimal expectations listed above.

Acknowledgements

Group Member Contributions

Ian: Loading Libraries and Data, Using R, Exploring Your Data

Kaitlin: Tutorial screenshots, Additions to Using R, Visualization with Graphs, t-tests, Linear Models, Markdown Formatting, GitHub

*Emily: Shapiro test, Log Transformation, p-value Discussion