STAT 140 - Lab 1

Introduction to R Markdown, Dataframes, and Data Visualization

YOUR NAME HERE

1/29/2020

Introduction to R Markdown

Within R Studio, we can write code in a script (this produces a .R file), as we did in Lab 0, or we can use an R Markdown file (produces a .Rmd file). While the former can be very useful, the latter is an excellent way to produce reproducible statistical analyses and generate self-contained reports. You can include text (as you would in Word, for example) and R code that produces data summaries, plots, and statistical analyses in the same document. For this reason, we will be using R Markdown in this class for labs and for many homework assignments. There is a bit of a learning curve involved in learning both R and R Markdown, so be patient, work together, and ask questions!

Inserting R chunks into R Markdown

Since Markdown allows us to combine text and code, we have to tell Markdown when text should be treated as R code. Everything we have done so far is simple text editting. In order to include code, we have to insert something called an R chunk. We will write all our code within R chunks (you can insert as many as you need).

How to insert an R chunk

1. At the top of your script pane (the pane in which you are currently working), you will see a green icon with a white c in it and a plus sign in the upper left corner. Next to it is the word "Insert" with a dropdown menu. Click on this.
2. The dropdown menu will contain several choices, the first of which is R. Click on R. This will insert an R chunk where your cursor was.

How to name an R chunk

It is good practice to name your R chunks - it makes error messages more informative. Each chunk must have a unique name. If two chunks have the same name, the knitting process will throw an error. In this example, the name of this R chunk is name1.

How to add other information to an R chunk

There are a number of arguments that you can add after you specify the name of your R chunk that will change how the R chunk prints in your final document. Put a comma after the name of your R chunk and type tidy=TRUE. This will be helpful when you write long lines of code later and want them to not run off the page of your knitted document. You should make a habit of including tidy=TRUE in your R chunks. This should make your life easier.

How to execute an R chunk

Depending on what you are doing, you may want to run just one or two lines, or you may want to run an entire R chunk or multiple R chunks. You can achieve these objects either through a point and click approach or by learning keyboard shortcuts. Here are four common things you might want to do:

1. Run selected lines: Use this if you just want to run a few lines, but not all the lines in a single R chunk. By default, if you run selected lines, this will execute the code on the same line as your cursor (just one line of code). You can also highlight more than one line and then run selected lines to run all highlighted code.
2. Run current chunk: This will run all lines of code in the R chunk where your cursor is.
3. Run all chunks above: This will run all lines in all R chunks above, but not including, your current R chunk (where your cursor is).
4. Run all chunks: This will run all lines in all R chunks in your Markdown file. If you run into error messages while knitting your document (more on this later), it is helpful to run all chunks to determine whether the bug corresponds to R code itself or R Markdown.

Point and click

At the top of your script pane (the pane in which you are currently working), you will see an icon that looks like a white screen with a blue dash and a green arrow pointing to the right. Next to this you will see the word "Run". If you click on Run, you will get a dropdown menu that will allow you to run selected lines, current chunks, etc.

Keyboard short cuts (recommended)

1. Run selected lines:

- PC: Ctrl+Enter;
- Mac: Command+Return

2. Run current chunk:

- PC: Ctrl+Shift+Enter;
- Mac: Command+Shift+Return

3. Run all chunks above:

- PC: Ctrl+Alt+P;
- Mac: Command+Alt+P

4. Run all chunks:

- PC: Ctrl+Alt+R;
- Mac: Command+Alt+R

Including comments in an R chunk

It is important to document what you are doing in your code so that either future you or someone else reading your code will understand it. While this may not seem that important now, it will become increasingly important as your programming become more involved.

A comment is a line in a code chunk that explains what is being done but that you do not want to run. For example, I may want to document that I am adding 2 and 3. The following is an example R chunk:

```
# Add 2 and 3 (This is a comment - the # keeps R from trying to read this as
# executable code.)
2 + 3
```

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

Notice that the pound sign is used to add a comment within the R chunk, but outside of the R chunk, pound signs let us specify headers of different sizes (depending on how many pound signs are used)!

Dataframe exploration

The data we will be using in this class will be stored as an object called a dataframe. You can think of this like an Excel spreadsheet - a dataframe has rows, columns, and column names. In this section, we will learn how to access a dataset that comes preloaded with R and several functions and commands you can use to learn about your data set and extract pieces of it.

Calling built-in datasets in R

We will be working with the Iris data set throughout the rest of this lab. Since this is a preloaded data set, we just have to call it. We use the function data() to do this. This must be done *before* we try to perform any functions on this data set.

```
## Load iris
data(iris)
```

Exploratory functions

head()

1. Insert an R chunk.
2. Name it head fcn. Also include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating head function".
4. Type a line of code that reads head(iris) and run it.
5. BRIEFLY summarize what head() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

summary()

1. Insert an R chunk.
2. Name it summary fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating summary function".
4. Type a line of code that reads summary(iris) and run it.
5. BRIEFLY summarize what summary() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

dim()

1. Insert an R chunk.
2. Name it dim fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating dim function".
4. Type a line of code that reads dim(iris) and run it.
5. BRIEFLY summarize what dim() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

class()

1. Insert an R chunk.
2. Name it class fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating class function".
4. Type a line of code that reads class(iris) and run it.
5. BRIEFLY summarize what class() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

nrow()

1. Insert an R chunk.
2. Name it nrow fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating nrow function".
4. Type a line of code that reads nrow(iris) and run it.
5. BRIEFLY summarize what nrow() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

ncol()

1. Insert an R chunk.
2. Name it ncol fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating ncol function".
4. Type a line of code that reads ncol(iris) and run it.
5. BRIEFLY summarize what ncol() does and write this summary as text (outside your R chunk). This can be a bulletted list, or a couple of sentences - whatever will be most helpful for you to reference later.

str()

1. Insert an R chunk.
2. Name it str fcn and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating str function".
4. Type a line of code that reads str(iris) and run it.
5. The function str() gives us a lot of information about the objext iris. It says that it is a dataframe ('data.frame':), and that there are 150 observations (150 rows) and 5 variables (5 columns). This information is followed by 5 lines starting with $, which give the five variables, the type of variable, and some information about what the entries for the variables look like. **We need to match R's language for variables with the language we have used in class. Here we have four variables of type 'num' and one of type 'Factor'. Using the language we learned in class, what kinds of variables are these?**

Extracting a single variable from a data frame

Sometimes we just want to look at a single variable (column) in a dataframe (I may refer to this as a vector). In this case, we will use the dollar sign to index into the dataframe. The general syntax is dataframe_name$column_name. Note, dataframe_name and column_name are being used as a general place holders here - you will have a specific dataframe_name and column_name that you will use.

1. Insert an R chunk.
2. Name it extract_var and include tidy=TRUE.
3. Inside your R chunk, add a comment that says, "Investigating $".
4. Type a line of code that extracts the Species variable from the iris data set and run it.
5. Write a quick note of what this returns. Later this will be useful if you want to take the mean of a single variable, for example, or perform other functions on a vector.

6. Add another comment that says, "Find the length of a vector".
7. We learned that dim(), nrow(), and ncol() will give us information about the size of a dataframe. Try applying these functions to your vector to determine its length. What happens? Why? (Think about what the error message is telling you.)
8. Now, replace the function you tried in 7. with length(). Does this work? Why? (Consult help documentation by typing ?length.)

Loading external packages

While R Studio comes preloaded with considerable functionality, sometimes we have to provide additional commands to load other functions. One of the great things about R is that it is open source, so people contribute to it frequently, expanding on the things that can be easily implemented in R. These contributions often come in the form of *packages*, which we have to install (the first time) and load during our R session. The first such package we will use is called ggplot2. This package is great for data visualization.

Installing a package for the first time

The first time you want to use an external package, you need to install it. This only needs to be done *once* if you are working locally on your computer. **If you are working on the server, the packages we will be using in this class are preinstalled and you can skip this step.**

```
# The following will install ggplot2. Currently it is commented out. If you
# need this line, just remove the # and then run it. After you install it,
# you should comment out the line again (replace the # at the beginning of
# the line).

# install.packages('ggplot2')
```

Loading an external package that is already installed

If you want to use an external package, you will have to load it each time you start a new R Session or R Markdown file. Usually this is done in a setup R chunk (an R chunk with the name setup). It looks like this:

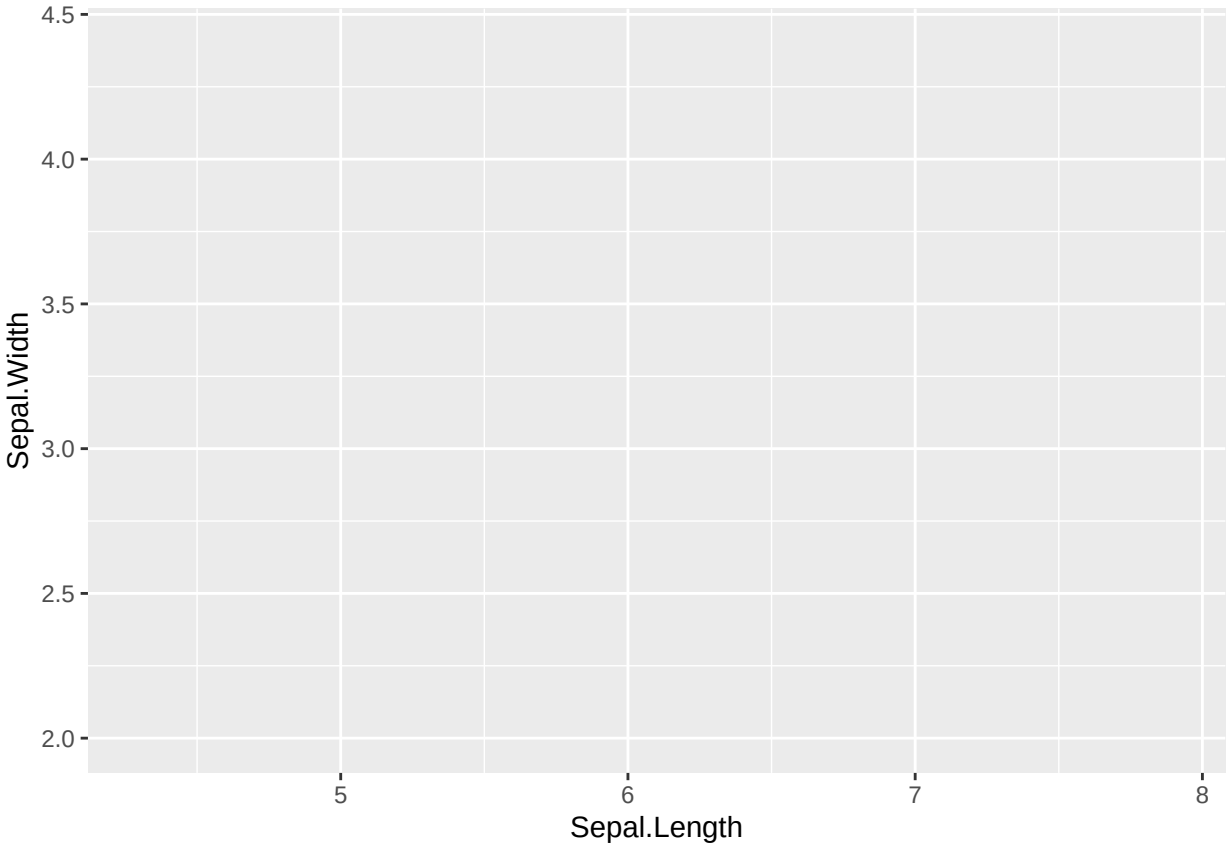
```
## Load packages
library(ggplot2)
```

Data visualization

We will be using ggplot2 in this class extensively. The functions in this package work by adding layers to a plot.

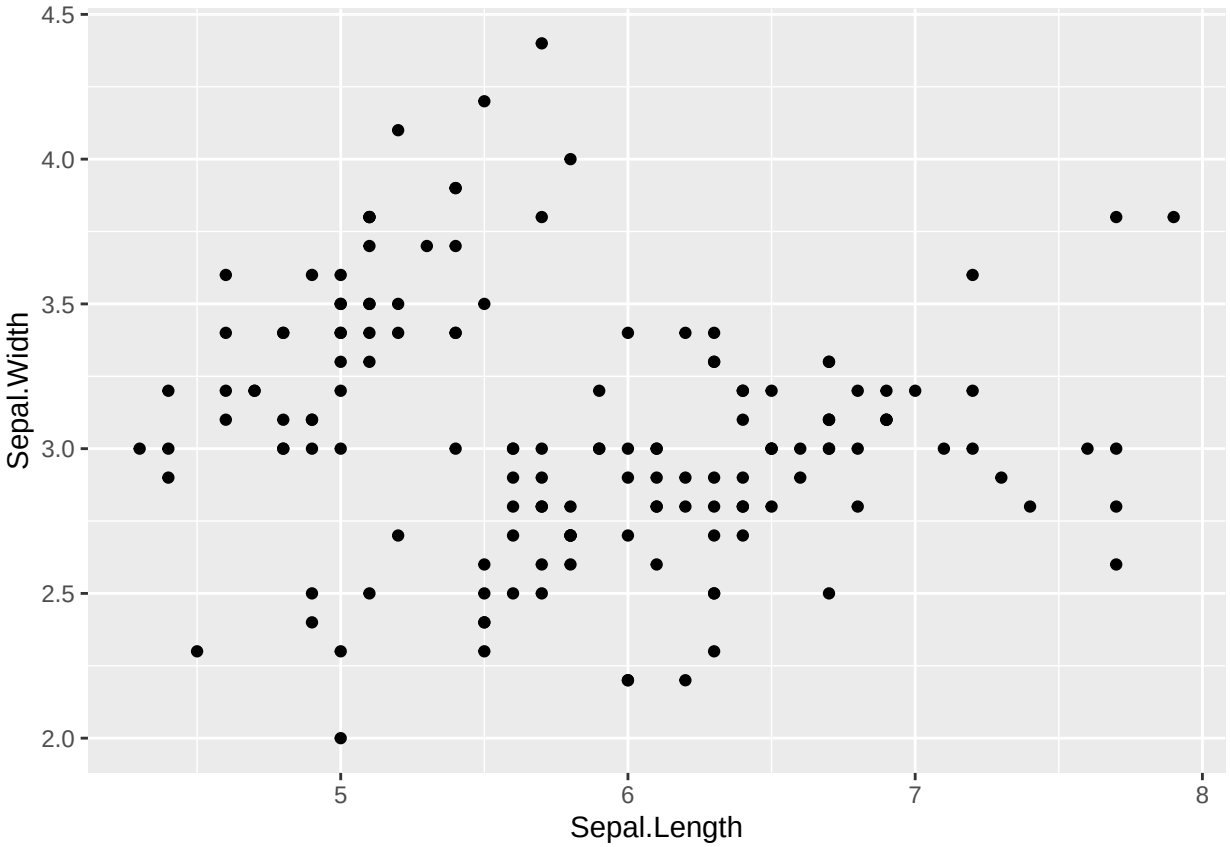
The following code specifies the data and the variables we want to plot. Try running it - does it produce a plot?

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



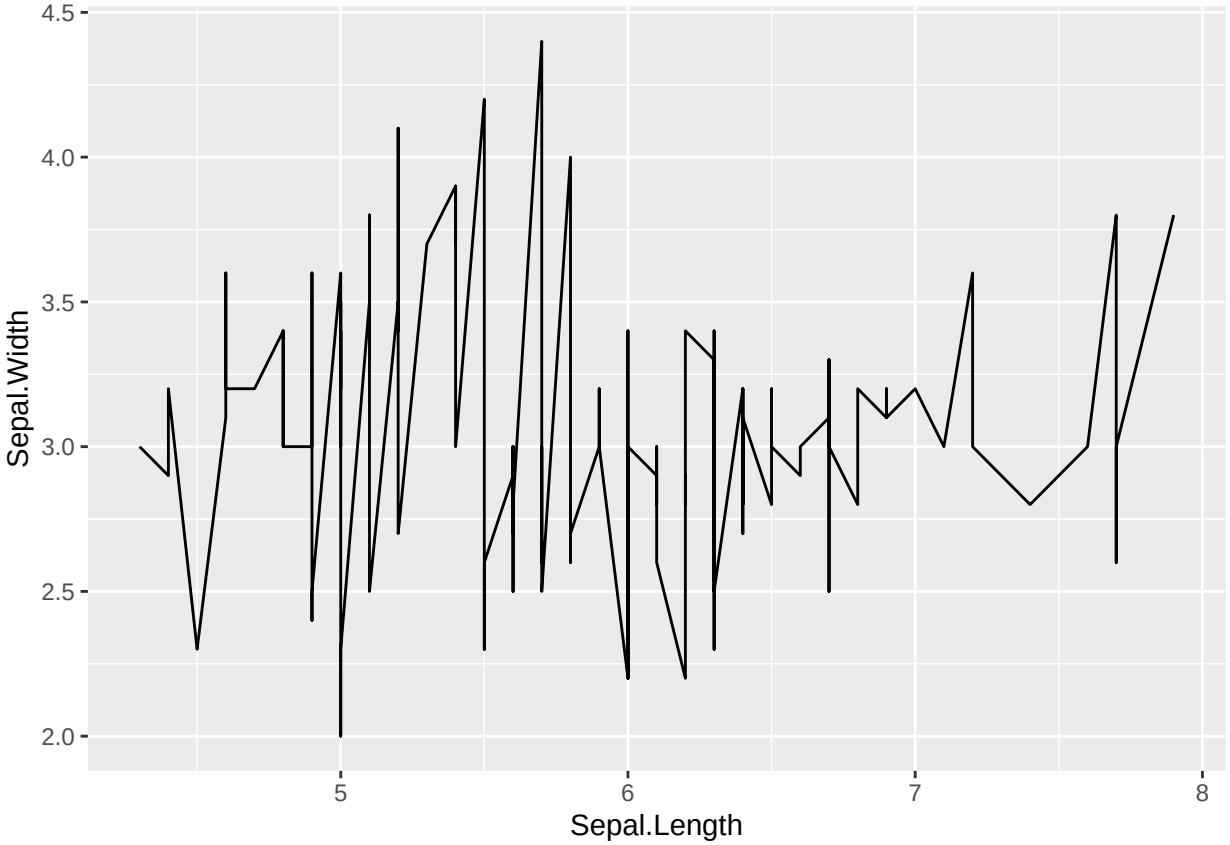
In this next chunk, we add another layer to the plot so we produce a scatterplot. We add geom_point() to the code.

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



If we wanted a line graph instead, then we can change the argument from geom_point() to geom_line(). Sometimes this kind of a plot makes sense, but in this case it's not particularly informative.

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



How to knit an R Markdown file

1. At the top of your script pane (the pane in which you are currently working), you will see a blue yarn ball and knitting needle icon. Next to it is the work "Knit" with a dropdown menu. If you click directly on the word Knit, rather than the arrow, an HTML document will be produced by default. There are other file formats to which we can knit, too, including PDF and Word (you can see these by clicking on the arrow). In this class, knitting to an HTML document should be sufficient.
2. **Debugging the knitting process.** Sometimes you will try to knit your document, and the process will produce an error. The error may be a result of a bug in your code (an R bug - less likely if you have been running your code and debugging as you go along before knitting) OR a problem with how something is formatted in Markdown (we'll call this a Markdown bug - generally more common). Throughout the semester, we will be compiling a list of bugs encountered and approaches to fixing them. This will be available as a Google sheet: INSERT GOOGLE SHEET INFO HERE. You should check this first if you encounter a bug in your code to see if there is a solution posted. If it is a new bug, you should add it to the sheet.