ST 411/511 Lab 1: Working with Data in R

Installing Required Packages

The first thing that you should do when you start a new R markdown document is to load the *R packages* that you'll need for your R session and to create your document. For this lab, and in general for this course, you'll want to load the **ggplot2** package and the **Sleuth3** package. *ggplot2* contains a collection of functions that allow for making sophisticated, publication-ready graphics. *Sleuth3* is the package that accompanies your textbook and contains data we will often be working with.

The *very first time* that you want to load a package on any computer, you have to first install that package. The following chunk of R code installs *ggplot2* and *Sleuth3*. To execute this code, click on the green triangle at the top right side of the chunk.

> I have commented out the install.packages commands in the following chunk (i.e., by putting a "#" sign in front of them). You should "uncomment" these commands the first time you run this R chunk, and then I recommend that you comment them out again. There is no need to keep re-installing an R package after you've done it once.

```
#install.packages("ggplot2")
#install.packages("Sleuth3")
```

You only need to install each package *once* ever.

To access the functions in *ggplot2* and the datasets in *Sleuth3*, you need to load these packages into your current R session using the `library()` function. You will need to re-load R packages during every R session. Go ahead and run the following chunk of R code:

```
library(ggplot2)
library(Sleuth3)
```

Remember that R is case sensitive, so using the command `library(sleuth3)`, for example, will result in an error message.

Using R like a calculator

You can use R to perform basic calculations. Try executing this chunk to see examples of addition and division.

> As you work through this lab, make sure to try executing every code chunk to look at the results.

```
1 + 2
```

```
## [1] 3
```

```
10 / 5
```

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

You can create a vector of numbers that R will save using the `c()` function. We then assign that we want that vector of numbners to be named `x` using the `<-` operator.

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Here, the vector x contains the values 1 through 10. Make sure that you execute this code chunk using the green arrow at the top right of the chunk so that it gets defined into the global environment.

What is the sum of the 10 values in x? We could find this two different ways:

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

```
## [1] 55
```

```
sum(x)
```

```
## [1] 55
```

We get the same answer, but clearly the second method using the `sum()` function is easier to type.

What is the average (mean) of the 10 values in x? Again, we could find this two different ways: min()

```
(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) / 10
```

```
## [1] 5.5
```

```
mean(x)
```

```
## [1] 5.5
```

The two answers are the same, but the `mean()` function is a lot easier (and now imagine if we had millions of data values..).

There are lots of other summary measures you may be interested in for your data. Try out a few of these:

- Median, using the `median()` function
- Maximum, using the `max()` function
- Minimum, using the `min()` function
- Variance, using the `var()` function
- Standard deviation, using the `sd()` function

  Practice making a code chunk and tyring out a few summary measures on the vector of numbers x.

```
median(x)
```

```
## [1] 5.5
```

```
max(x)
```

```
## [1] 10
```

```
min(x)
```

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

```
var(x)
```

```
## [1] 9.166667
```

```
sd(x)
```

```
## [1] 3.02765
```

Data Frames

Data can have different structures in R. We just looked at a vector (x) of length 10. This was a way to represent one variable.

Another common data structure in R is a **data frame**, which contains information about multiple variables. Each column is a different variable.

Consider the **mtcars** data set in R, which contains data about fuel consumption and other aspects of a variety of cars. You can view this data in a separate window using the following command (after removing the '#'):

```
#View(mtcars)
```

(The `View()` function will display the data in a spreadsheet, like you might be used to seeing in Excel. This can be useful for small datasets, but slow or cumbersome for large datasets. When you Knit a document you also don't want to print out all your data, which is why we comment out that line of code.)

Each row is a different car. The first column, **mpg**, contains the miles per gallon each car gets. The second column is a different variable, the number of cylinders.

We can examine properties of the data frame in R. How big is the data frame?

```
dim(mtcars)
```

```
## [1] 32 11
```

It has 32 rows (cars), and 11 columns (variables).

What are the names of the variables?

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Perhaps I don't know what these abbreviations mean. I can look up documentation included with the *mtcars* data set using:

```
?mtcars
```

Documentation will pop up in the *Help* tab of a different pane. We can see that, for example, *carb* stands for the number of carburetors.

Suppose we want to find the average **mpg** for the cars in this data frame. We need to tell R both the name of the data frame to look in and which column we are considering. Here are two ways to find the average **mpg** for cars in this data frame:

```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

```
mean(mtcars[,1])
```

```
## [1] 20.09062
```

In the first method, we are calling the **mpg** column by name: the $ operator followed by the column name is used to reference the column we are interested in. The second method references **mpg** by its position: it is the first column. If we wanted to find the average number of cylinders instead, we would use `mtcars[,2]`.

Try finding the average weight (the **wt** column) for cars in this data frame.

```
mean(mtcars$wt)
```

```
## [1] 3.21725
```

```
mean(mtcars[,6])
```

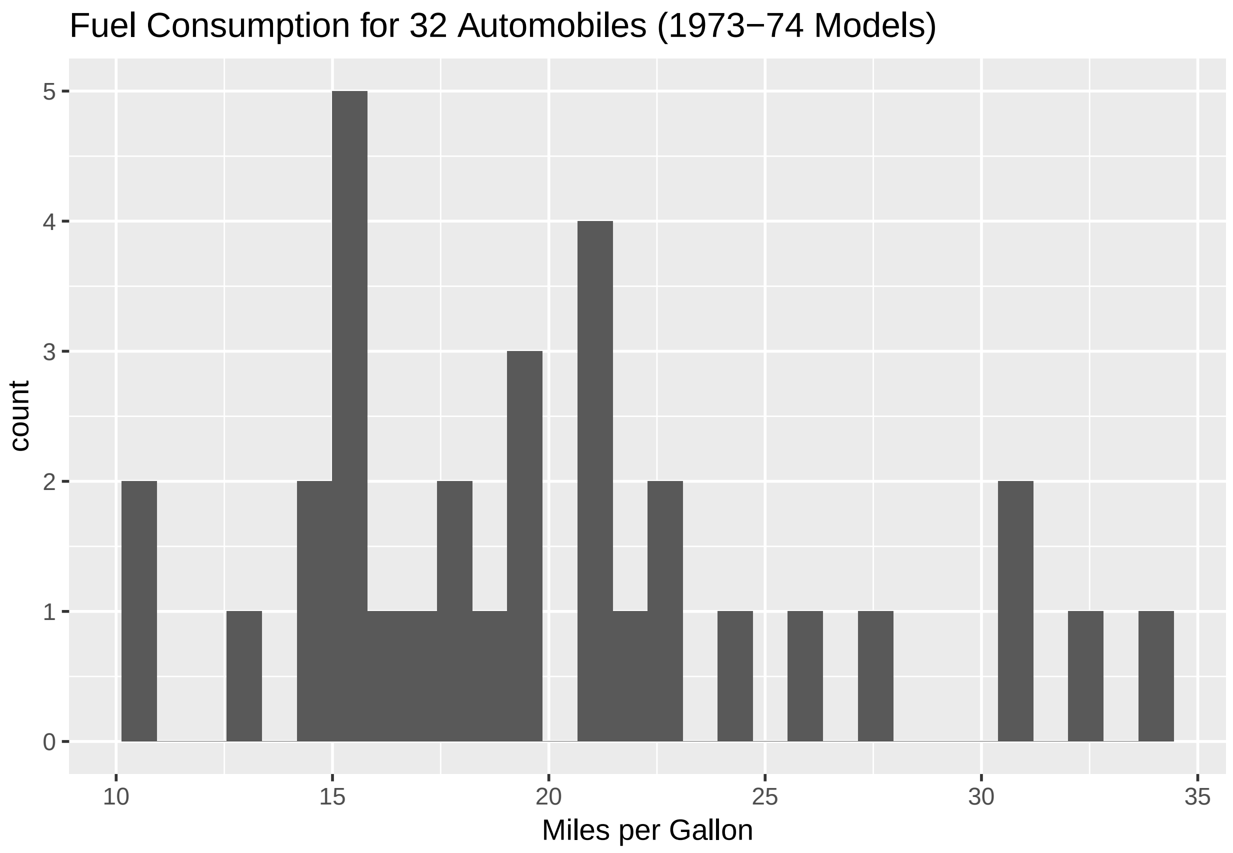
```
## [1] 3.21725
```

A Histogram using ggplot

Run the following chunk of code to obtain a histogram of the miles per gallon (**mpg**) from the **mtcars** data.

```
ggplot(data = mtcars, aes(mpg)) +
  geom_histogram() +
  xlab("Miles per Gallon") +
  ggtitle("Fuel Consumption for 32 Automobiles (1973-74 Models)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Notice first of all that we get a suggestion that we pick a better binwidth. More on this below.

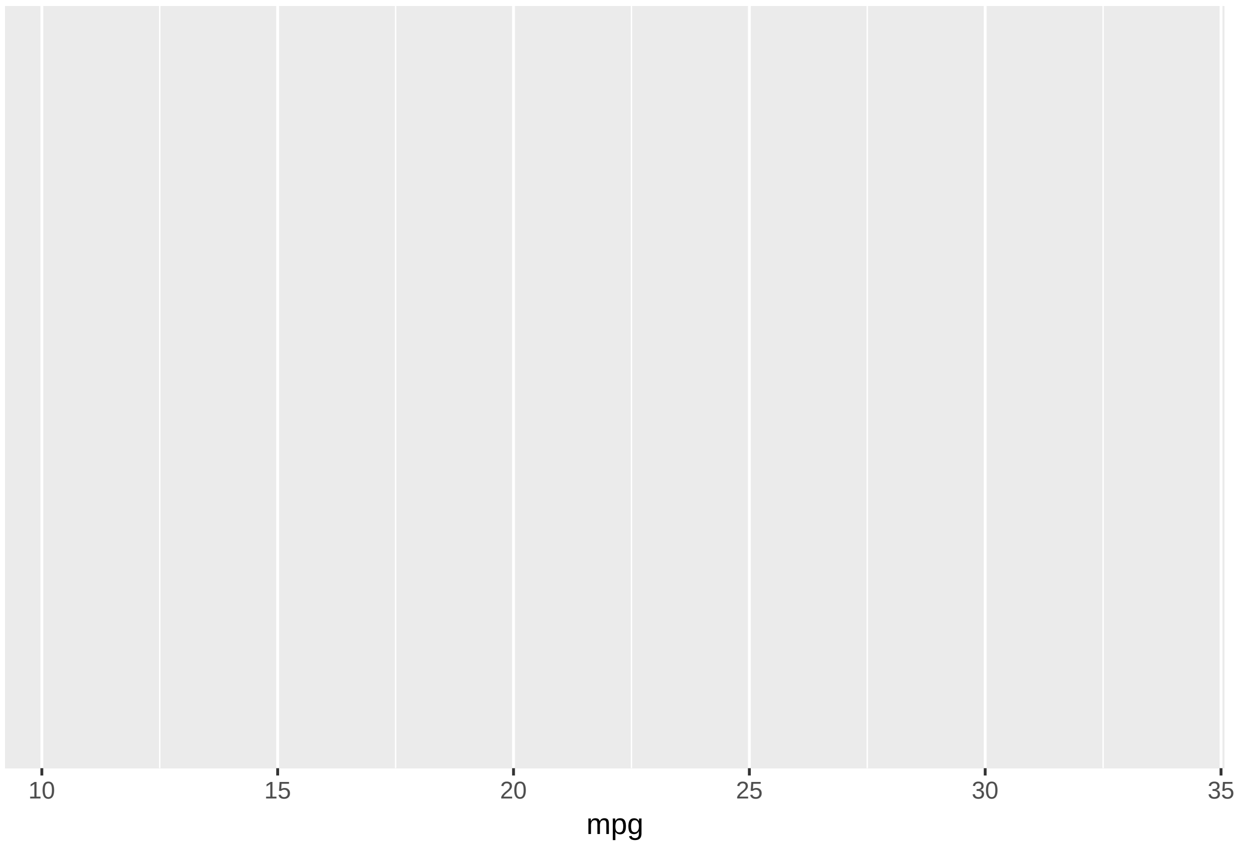
The main function in the *ggplot2* package is, not surprisingly, called **ggplot** (for those of you who are interested, "gg" stands for "Grammar of Graphics," which is the framework upon which the *ggplot2* package was built).

We use the *ggplot* function to identify the data frame that contains the variable(s) that we want to plot. In this case, I used the argument `data = mtcars` to tell *ggplot* to use the *mtcars* data frame.

The other argument that I passed to *ggplot* is the `aes` element, which stands for "aesthetic element." This is how I tell *ggplot* which variable(s) from the data frame that I want to plot – in this case it's just the *mpg* variable.

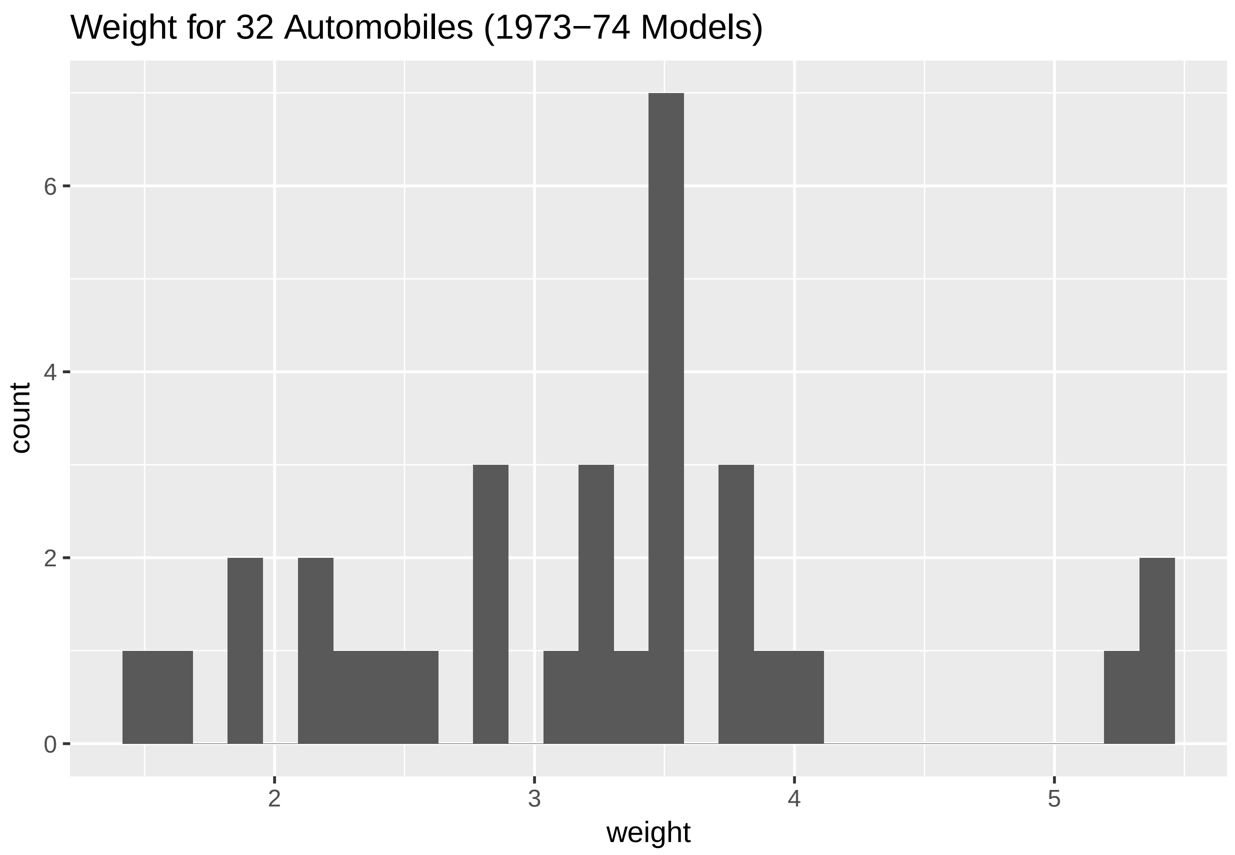
Now, if I just use the *ggplot* function, without anything else, you will get a blank plot. You can try this here:

```
ggplot(data = mtcars, aes(mpg))
```



```r
ggplot(data = mtcars, aes(wt)) +
  geom_histogram() +
  xlab("weight") +
  ggtitle("Weight for 32 Automobiles (1973-74 Models)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



“‘

The reason for this is that we haven’t yet indicated what type of plot we want – is it a scatterplot, a histogram or some other type of plot? In this case I want to plot a histogram, so I added the command `+ geom_histogram()` to the *ggplot* command. This tells R that I want a histogram.

The syntax of the *geom_histogram* command, and other commands like it (*geom_point*, *geom_line*, etc.) is that these commands tell R what *type* of plot, or **geom**etric object, you want to create.

The other two commands used are `xlab`, which places a label on the x-axis, and `ggtitle`, which puts a title on the plot.

For completing homework 1, I recommend that you look at the help file for *geom_histogram* to see how you might change the bin width. It is selected automatically, but in this example R has given the warning that we may want to select it manually. There are other great resources for *ggplot2* that you can access by typing *?ggplot2* at the command line prompt in the Console pane.

A good reference for *ggplot2* is here: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

> WARNING: I have found there to be a steep learning curve for ggplot, more than for base graphics in R. It’s worth the climb, however, because you’ll end up being able to produce some really beautiful and informative plots.

For the purposes of this class, the TAs and I will provide you with the basics of all functions and R information that you will need to fulfill the requirements of the course. That said, I really encourage you to go beyond the basics! Also, please don’t hesitate to post questions to the discussion boards on Canvas.