An Introduction to R

Learn the Basics

Joey Stanley
Doctoral Candidate in Linguistics, University of Georgia
joeystanley.com
orcid.org/0000-0002-9185-0048

Presented at the UGA Willson Center DigiLab
Wednesday, September 13, 2017

This is the first installment of the R workshop series. This document will cover some of the basics of R including the following topics: (1) what R is and what some of its alternatives are, RStudio, and installation; (2) R Basics, such as using R as a calculator, variables, and basic functions; (3) Getting data into R from a .csv, .txt, or other data types; (4) working with, displaying, extraction portions of, and filtering your data, with tangents on logical operators and R packages; (5) basic visualizations; (6) where to go for help, both in R and on the internet.

Download this PDF from my website at
joeystanley.com/r

(Updated September 12, 2017)

1 Introduction

1.1 What is R?

At its core, R is an open source programming language for statistical computing. Let's break this statement down a little bit.

It's *open source*. This means that everything about it is free and open to the public. Open source software is in response to proprietary software which are owned by some entity, usually cost money, and have things like their source code hidden from the public. I won't get into the politics between these two, but one positive aspect of open source stuff is that other people can contribute to it. R is an excellent example of a community-driven effort to make something better: there are thousands of user-submitted add-ons, called "packages" or "libraries" that you can download to enhance your R code. We'll get to that later.

Second, R is a *programming language*. I won't get into the technicalities of what that means. For our purposes, it's sufficient to just know that R is akin to other languages like Python, C#, C++, Java, PHP, Perl, Swift, Ruby, etc. So if you've never coded before, there are some new concepts to learn, but if you have, there's a lot of cross-over information that'll come naturally.

Finally, R is for *statistical computing*. Each programming language has its strengths and weaknesses and no programming language does everything equally well. R's strength is working with data and running statistical analyses. This means isn't not as good as working with text as Perl and not as good as making stand-alone software as Java or C#. But when it comes to data analysis, R is a workhorse, and will get the job done.

Basically, with R you'll be able to analyze your datasets more efficiently than many other alternatives, and with the help of user-submitted libraries, you'll be able to do some pretty neat stuff with it.

1.2 What are some alternatives to R?

Say you don't like R. Maybe you don't want to bother trudging through another programming language or you had an ex who's name starts with R. Luckily for you, there's some alternative software you can use instead.

- *SPSS* is a common alternative in the humanities. Unfortunately it only runs on Windows, so sorry, Mac users. Also, it's proprietary software.
- *SAS* might be the way to you if you're in the sciences. Again though, proprietary software.
- If you're in economics or epidemiology, you might be more at home using *Stata*. You guessed it though, it's proprietary software.

- Maybe you come from a mathematics or engineering background and want to stick with *MATLAB*. Well, it too is proprietary software.
- If you don't even want to bother with coding, you might like JMP. The downside is… do I even have to say it anymore?

Those of you with a keen sense of discernment might notice that all these alternatives have something in common. The fact that they're proprietary software means that it costs money to use them. I've got nothing against proprietary software. I'm just poor. Most students can get access to some or all of these through their university's license, so by all means, go ahead and use these others (they are all very good). But as soon as you graduate, you'll need to fork over a ton of money or hope you have a licence wherever you go.

R is free. You download it to your computer and it's there forever. No need to renew site licenses. No need to put in credentials. No annoying "Upgrade to premium to access these features!" messages. It's free and yours forever. Above all else, this is my favorite part of R.

(I will say though, Python is also very good. It's also free and has tons of user-submitted packages. I just haven't had the opportunity to learn it yet.)

1.3 R vs. RStudio

If you're just getting started with R, you may have heard of something RStudio and you might be wondering what it is.

R is the programming language. It comes standard on many computers (you can download it *here* if you don't have it). You can run R commands through it and it works fine. However, it's hard to work with for several reasons. You have no way of keeping track of your script, so if you stop working in the middle of something, when you start up again you'll need to type all those commands over again. It's also hard to keep track of your objects or variables that you've created in a particular session. It's harder to access help and see visuals as well. Again, it works fine, it's just a little difficult.

RStudio is a good solution to these problems. It's another open source piece of software that acts as a shell around R. At its core, all it does is run the R language. But it makes it a lot easier for you. You can create R scripts so you don't have to type every command every time. It keeps track of your history and what variables you have currently. It also makes it a lot easier to access help, see visuals, and lots of other stuff.

On top of that, RStudio has a lot of other pretty neat things that, as far as I know, don't work in base R. For example, you can create HTML documents (like the one you're reading right now) directly in RStudio! You can make PDFs and Word files too, with R code, output, and graphics built-in. This is called *R Markdown*. You can also make interactive webpages using *Shiny*, which allow users to interact with your data. Later in this R Series (probably Spring 2018), I'll be giving workshops on both of these.

The team that works at RStudio is actively involved in making really handy libraries. For example, Hadley Wickham, RStudio's Chief Scientist is the author of packages like `dplyr`, `ggplot2`, and `tidyr`, which are among the language's most downloaded libraries. Later this semester, we'll have a workshop on *ggplot2* (on October 12, 2017) and another on the rest of the *tidyverse* library (on November 11, 2017) which includes `dplyr`, `tidyr` and many other super awesome packages. (That's right, they have their own websites because they're really that cool!)

1.4 Installing R and RStudio

Let's get down to business. To download R, go to *https://cran.r-project.org/mirrors.html*. This will take you to a list of CRAN mirrors. All these lists of sites are identical, they're just hosted on various servers across the world to handle the traffic. Just pick one near your current location and click on it. From there, download the package appropriate for your computer.

Mac users will be taken to a screen where they'll give you various versions to choose from. At the time of writing, the latest package is 3.4.1, so go ahead and download that one and install it like any other piece of software. Windows users will have a link that says "install R for the first time" which will take them to the download page. You can then install R like normal.

To download RStudio, go to *https://www.rstudio.com* and click "Download" under RStudio. There are several intense versions of RStudio and we only need the free Desktop version, which is the one furthest to the left. Click the download button and then click on the link appropriate for your operating system.

That's it. Pretty straightforward for both of these. Once you have both of them installed, you'll only ever need to run RStudio from here on out.

2 R Basics

Open RStudio and create a new script by going to `File > New File > R Script`. This will open what looks like a text editor in the top left portion of your RStudio screen. This looks like a text editor because that's actually all it is. R scripts are just text files with nothing fancy about them. When you save them, instead of .txt they'll get a .R appendix, which lets your computer know to run it as an R script.

2.1 R as a calculator

Obligatorily, any tutorial on R has to show you that R can be a calculator. In your R script, type the following:

```
2+2
```

With your cursor still on this line, hit command+enter (or for Windows, ctrl+enter), which is the keyboard shortcut for "execute this line". In the bottom left quadrant of RStudio, you'll now see a new line, in blue `>2+2` and then, in black, the output `[1] 4`.

```
## [1] 4
```

You've just executed your first R command! You can think of the bottom left quadrant, the *console*, as the actual R portion of RStudio. The script above it is just a placeholder for the various commands. When you want to run a command, RStudio will send it down to the console and execute it. Once it does that, the next line in the console will be the output, which in this case is `4`. The `[1]` before it just tells you that the output is actually a list that is one unit long. No need to worry about that right now.

You can do other arithmetic in R as well:

```
10*(5-2)/3+1

## [1] 11
```

Not too shabby. But this is boring. Let's move on to bigger and better things.

2.2 Variables

You can create *variables* to store data. Variables have arbitrary names, so you can call them whatever you want. To create a variable, provide it's name (in this case `six`), then the assignment operator `<-`, and some value. (Mac users: a keyboard shortcut for the assignment operator is option+dash.) Let's create a variable called `six` and give it the value of `6`.

```
six <- 6
```

Okay great. Now what we can do is use this `six` variable as if it were any other number.

```
six + 3

## [1] 9

six * 2

## [1] 12

1 / six

## [1] 0.1666667
```

Just to show the names really are arbitrary, you can use whatever names you want and it'll still work.

```
blue <- 4
six + blue

## [1] 10
```

```
seven <- 8
dog <- -5
blue * seven + dog

## [1] 27
```

It's probably not a good idea to use these kinds of variable names—and in general it's good practice to give brief but descriptive variable names—but it goes to show you can use whatever names you want.

You can store multiple numbers in a single variable using the `c()` command, which stands for "combine". Let's create a new variable called `fibs` and have it contain the first several numbers of the Fibonacci sequence.

```
fibs <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144)
```

Technically this list of numbers is called a *vector* in R, specially, a *double vector*. This doesn't mean that there's anything repeated or doubled: "double" is a term used in computer science to essentially mean number. What we have is a vector of numbers. What's cool about these is that you can treat it like a single number, and it'll run the command on each element of the vector.

```
2 + fibs

##  [1]   2   3   3   4   5   7  10  15  23  36  57  91 146

six * fibs

##  [1]   0   6   6  12  18  30  48  78 126 204 330 534 864
```

You can display the entire contents of the vector by simply typing the name of the variable.

```
fibs

##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144
```

But if you want to access a single item in that list, type the variable name and in square brackets immediately after the variable name, type what the which element you'd like to access. For example, here's how you would access the first, fifth, and tenth elements of the list

```
fibs[1]

## [1] 0

fibs[5]

## [1] 3

fibs[10]

## [1] 34
```

You can even pass in a list of numbers! If you want to see all three of these elements of the list at once, just wrap them up in the `c` function and put that in place of the number.

```
fibs[c(1,5,10)]

## [1]  0  3 34
```

These lists are really important, because when it comes time to read in your own spreadsheets into R, each column will be treated as a list just like this one. We'll get to that in a sec.

2.3 Arithmetic Functions

This is great and all. We can do more though. Here, let's look at what are called *functions*. These are commands that take one or more *arguments*. The function takes these arguments, works its magic under the hood, and returns some value. For example, the `sqrt` function takes the square root of some number.

```
sqrt(16)

## [1] 4
```

Note that the functions are case sensitive, and the arguments go in parentheses. If there are multiple arguments, they are separated by commas. Some other functions include `sum`, `mean`, and `range`. You can pass variables as arguments, and you can even nest functions.

```
sum(six, 3, dog)

## [1] 4

mean(fibs)

## [1] 28.92308

range(fibs, -4, sqrt(16), six)

## [1]  -4 144
```

You should get very comfortable running functions and getting all the syntax right like keeping track of your parentheses, commas, and spelling. The idea that you run functions, and on functions within functions, is super common in R. You don't have to memorize all the functions that R has, and in fact, you'll probably never use them. You're probably wondering what all the functions in R even are and how you're supposed to know about them. Google. There is tons of documentation for R online, and you'll hopefully be able to find your answer very quickly.

3 Getting data into R

At this point, a lot of R tutorials start you off with working with generating data within R itself. While this is an important skill to learn eventually, what's most relevant to you right now is getting your own data into R.

I'm going to assume you have your own spreadsheet somewhere saved on your computer. Ideally, each row represents one *observation* and each column is a variable of that observation. For example, if you had a spreadsheet that had the area, population, and capital of each of the 50 states, you'll have 50 rows and 4 columns (one for each of the variables and the fourth for the state name). Presumably, your data is clean and tidy, meaning that dates and numbers are formatted the same, capitalization is standardized, and everything is consistent. I could go on for a long time about the importance of making your data clean, but suffice it to say that it's paramount for proper analysis in R.

The functions for reading in data in R depend on what kind of file you have. The most common options are `.csv`, `.txt`, and an Excel file. Let's look at each one of those.

3.1 .csv files

If your file has *comma-separated values* (it ends with .csv), that's the easiest way to go. You can use `read.csv`, and as the only argument, put the full *path* to the file you want to read in, in quotation marks. For example, I've got a file on my Desktop called `menu.csv`. This file contains all the menu items at McDonald's with complete nutrition information and is available for free at *Kaggle.com*. To read this file in, the function would be this:

```
read.csv("/Users/joeystanley/Desktop/menu.csv")
```

If I were on a Windows, it might be something like this:

```
read.csv("C:\\Users\\joeystanley\\Desktop\\menu.txt")
```

Note that Mac users should use single forward slashes while Windows users have to use double back slashes.

To execute this command, I would put my cursor anywhere on the line, and either click the Run button on the top left of the R script, or preferably, use the keyboard shortcut `command+return` for Macs or `control+enter` for Windows.

When you do this, you'll start to see the contents of your file displayed in your R console (the bottom left portion of the RStudio screen). Hooray! You just read your data into R!

Unfortunately, all it did was read it in and forget it. Computers do *exactly* as they're told. What you didn't tell R was to *remember* the contents of the file. So, let's create a new variable called `menu` and save the contents of the `menu.csv` file into that variable.

```
menu <- read.csv("/Users/joeystanley/Desktop/menu.csv") # for Macs
menu <- read.csv("C:\\Users\\joeystanley\\Desktop\\menu.txt") # for Windows
```

Okay, so now we have a new `menu` object that has the full contents of my file. Before we move on to working with that file, let's see how to read in files that are in other formats.

3.2 .TXT FILES

If your file is a *tab-delimited* file (it ends with .txt), then you can use `read.table` instead. As an additional argument (which is separated from the path name with a comma), you may need to specify that the cells of your table are separated by tabs. To do this, add the `sep="\t"` argument (`\t` is "computer-talk" for a tab). Also, your file may have a header, meaning the first row of your file might contain the names of the columns. By default, `read.csv` assumes this, but for `read.table` you'll need to make that explicit so R knows what to do with them. You can add this using the `header=TRUE` argument. So the final command might look like this.

```
menu <- read.table("/Users/joeystanley/Desktop/menu.csv", sep="\t", header=TR
UE) # for Macs
menu <- read.table("C:\\Users\\joeystanley\\Desktop\\menu.txt", sep="\t", hea
der=TRUE) # for Windows
```

3.3 EXCEL FILES

If you data is in an Excel file, there are ways to get it into R. For now, the easiest solution is to simply save it as a `.csv` file and read it in that way. In November, when we talk about the `tidyverse`, we'll look at how to read in data from Excel.

3.4 DON'T LIKE TYPING PATH NAMES?

There is one other way to read in data that doesn't involve typing those long path names. You can use the `file.choose` command instead. When you do this, a window will open up and you'll be able to find your file and click on it just like you were opening any other file.

```
menu <- file.choose()
```

I don't like this option mostly because it just takes too many clicks. Every time I want to run this line of code, it takes 5 or so clicks to get the file I want. This gets really tedious. It's worth the time to just type (or copy and paste!) the path name to the file one time, and with a single keystroke you can load it in nearly instantaneously the exact same way every time.

4 WORKING WITH YOUR DATA

Okay, so your data is in R. Now we need to be able to view it to make sure it's all there, and also be able to extract portions of it.

4.1 Displaying your data

The easiest way to display your data is to simply type the name of the variable itself. However, depending on the size of your data frame, this could get huge. I've truncated mine, particularly the description so it would fit on this page, but yours might be different.

```
menu
```

```
##     Category                       Item  Oz Calories Fat Sugars
## 1  Breakfast               Egg McMuffin 4.8      300  13      3
## 2  Breakfast          Egg White Delight 4.8      250   8      3
## 3  Breakfast           Sausage McMuffin 3.9      370  23      2
## 4  Breakfast Sausage McMuffin with Eg 5.7      450  28      2
## 5  Breakfast Sausage McMuffin with Eg 5.7      400  23      2
## 6  Breakfast      Steak & Egg McMuffin 6.5      430  23      3
## 7  Breakfast Bacon, Egg & Cheese Bisc 5.3      460  26      3
## 8  Breakfast Bacon, Egg & Cheese Bisc 5.8      520  30      4
## 9  Breakfast Bacon, Egg & Cheese Bisc 5.4      410  20      3
## 10 Breakfast Bacon, Egg & Cheese Bisc 5.9      470  25      4
## 11 Breakfast Sausage Biscuit (Regular 4.1      430  27      2
## 12 Breakfast Sausage Biscuit (Large B 4.6      480  31      3
## 13 Breakfast Sausage Biscuit with Egg 5.7      510  33      2
## 14 Breakfast Sausage Biscuit with Egg 6.2      570  37      3
## 15 Breakfast Sausage Biscuit with Egg 5.9      460  27      3
## 16 Breakfast Sausage Biscuit with Egg 6.4      520  32      3
## 17 Breakfast Southern Style Chicken B 5.0      410  20      3
## 18 Breakfast Southern Style Chicken B 5.5      470  24      4
## 19 Breakfast Steak & Egg Biscuit (Reg 7.1      540  32      3
## 20 Breakfast Bacon, Egg & Cheese McGr 6.1      460  21     15
```

This may spill over onto multiple lines. That's only because my screen isn't wide enough to display the full row. An alternative that is easier for scrolling, is to use the `View` function (yes, that's a capital *V*). This opens up a new tab in RStudio, and you'll be able to view your data like you would a normal spreadsheet.

```
View(menu) # Not run here because we're not in RStudio.
```

You can also just display portions of your data using `head` and `tail`, which, respectively, show the first and last couple of rows.

```
head(menu)
```

```
##    Category                       Item  Oz Calories Fat Sugars
## 1 Breakfast               Egg McMuffin 4.8      300  13      3
## 2 Breakfast          Egg White Delight 4.8      250   8      3
## 3 Breakfast           Sausage McMuffin 3.9      370  23      2
## 4 Breakfast Sausage McMuffin with Eg 5.7      450  28      2
## 5 Breakfast Sausage McMuffin with Eg 5.7      400  23      2
## 6 Breakfast      Steak & Egg McMuffin 6.5      430  23      3
```

```
tail(menu)

##                Category                       Item   Oz Calories Fat Sugars
## 255 Smoothies & Shakes McFlurry with M&M\x89ûªs Can  7.3      430  15      5
9
## 256 Smoothies & Shakes   McFlurry with Oreo Cooki 10.1      510  17     64
## 257 Smoothies & Shakes   McFlurry with Oreo Cooki 13.4      690  23     85
## 258 Smoothies & Shakes   McFlurry with Oreo Cooki  6.7      340  11     43
## 259 Smoothies & Shakes   McFlurry with Reese's Pe 14.2      810  32    103
## 260 Smoothies & Shakes   McFlurry with Reese's Pe  7.1      410  16     51
```

4.2 Extracting portions of your data

Before, when we had several numbers saved into a single variable, we called it a *vector*. Now, we have an entire spreadsheet saved into the `menu` variable. Each column in your data frame is treated as a vector, and each row of that column is an element in that list. We call this type of variable a *dataframe*.

You can think of a vector as a one-dimensional variable and a data frame as a two-dimensional variable. To access an element of a vector, you type one number in the square brackets (`fibs[1]`). Because a data frame is two-dimensional, you'll have to send two numbers, separated by a comma: the first for the row number and the second for the column number.

```
head(menu) # The head() function displays the first few rows of a dataframe.

##    Category                     Item  Oz Calories Fat Sugars
## 1 Breakfast              Egg McMuffin 4.8      300  13      3
## 2 Breakfast         Egg White Delight 4.8      250   8      3
## 3 Breakfast          Sausage McMuffin 3.9      370  23      2
## 4 Breakfast Sausage McMuffin with Eg 5.7      450  28      2
## 5 Breakfast Sausage McMuffin with Eg 5.7      400  23      2
## 6 Breakfast      Steak & Egg McMuffin 6.5      430  23      3

menu[4,2]

## [1] "Sausage McMuffin with Eg"
```

The line `[1] Sausage McMuffin with Egg` shows the content of the element you just extracted. The `260 Levels: 1% Low Fat Milk...` just shows that there are 260 different items in that column and it goes ahead and list them in alphabetical order for you. You can ignore that for now.

So that's how you would extract a single cell in your spreadsheet. You can extract an entire row by leaving off the column number.

```
menu[3,]

##    Category             Item  Oz Calories Fat Sugars
## 3 Breakfast Sausage McMuffin 3.9      370  23      2
```

Notice how the output shows the column names along the top as well as the row number on the left side. You can extract an entire column by leaving off the row number. (I've truncated my list for display purposes: yours will be a lot longer.)

```
menu[,2]

##  [1] "Egg McMuffin"                "Egg White Delight"
##  [3] "Sausage McMuffin"            "Sausage McMuffin with Eg"
##  [5] "Sausage McMuffin with Eg" "Steak & Egg McMuffin"
##  [7] "Bacon, Egg & Cheese Bisc" "Bacon, Egg & Cheese Bisc"
##  [9] "Bacon, Egg & Cheese Bisc" "Bacon, Egg & Cheese Bisc"
```

As an alternative to extracting an entire column, it's easier to refer to the column by its name (`Item`) rather than its number (`2`). This is especially true if you have many columns in your spreadsheet and you don't feel like counting all of them. You can refer tho the column using the dollar sign `$`, which is placed between the variable name and the column name.

```
menu$Item

##  [1] "Egg McMuffin"                "Egg White Delight"
##  [3] "Sausage McMuffin"            "Sausage McMuffin with Eg"
##  [5] "Sausage McMuffin with Eg" "Steak & Egg McMuffin"
##  [7] "Bacon, Egg & Cheese Bisc" "Bacon, Egg & Cheese Bisc"
##  [9] "Bacon, Egg & Cheese Bisc" "Bacon, Egg & Cheese Bisc"
```

4.3 Filtering your data

We can also perform queries on your data so that we can filter your data in whatever way we want. We can view entire portions of the data frame just as easily in Excel as we can in R, so why bother with R in the first place?

In Excel it's certainly possible to filter your data. But it gets a little cumbersome if you have multiple filters on at once. Or if you have to switch back and forth between two or more different filters, you have to do a lot of clicking. With R, it's a little bit easier.

4.3.1 *Tangent: Logical Operators*

To use filters, I'll have to introduce a couple ways to *evaluate* data. What a filter does is it looks at your data, and decides whether certain rows meet certain conditions. If we start simple, and take our `fibs` vector, we can see how many of those numbers are greater than 10.

```
fibs

##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144

fibs > 10

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE
```

Here, the first 7 items in our list are `FALSE`, meaning they are not greater than 10. The last 6 are `TRUE`, so they are greater than 10. Similar to greater than, which uses the rightward-pointing angled bracket, we can use the following in similar ways:

- `>` means "greater than"
- `<` means "less than"
- `>=' means "greater than or equal to"
- `<=` means "less than or equal to"
- `==` means "equal to". Note here that you need *two* equals signs, and not just one.
- `!=` means "not equal to"

Here are some examples.

```
fibs > 10

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE

fibs < 20

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [12] FALSE FALSE

fibs >= 55

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [12]  TRUE  TRUE

fibs <= 13

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [12] FALSE FALSE

fibs == 1

##  [1] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE

fibs != 34

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [12]  TRUE  TRUE
```

4.3.2 Back to the data

We can use the same kind of thing to subset your data. For example, if we want to find all menu items that have 150 calories or less, we can do so like this.

```r
menu[menu$Calories <= 150,]
```

```
##           Category                      Item   Oz Calories Fat Sugars
## 39       Breakfast                Hash Brown  2.0      150 9.0      0
## 85          Salads Premium Bacon Ranch Sala  7.9      140 7.0      4
## 88          Salads Premium Southwest Salad   8.1      140 4.5      6
## 100 Snacks & Sides         Kids French Fries  1.3      110 5.0      0
## 101 Snacks & Sides                Side Salad  3.1       20 0.0      2
## 102 Snacks & Sides              Apple Slices  1.2       15 0.0      3
## 103 Snacks & Sides  Fruit 'n Yogurt Parfait  5.2      150 2.0     23
## 106       Desserts    Oatmeal Raisin Cookie  1.0      150 6.0     13
## 107       Desserts      Kids Ice Cream Cone  1.0       45 1.5      6
## 111      Beverages Coca-Cola Classic (Small 16.0      140 0.0     39
## 114      Beverages Coca-Cola Classic (Child 12.0      100 0.0     28
## 115      Beverages         Diet Coke (Small) 16.0        0 0.0      0
## 116      Beverages        Diet Coke (Medium) 21.0        0 0.0      0
## 117      Beverages         Diet Coke (Large) 30.0        0 0.0      0
## 118      Beverages         Diet Coke (Child) 12.0        0 0.0      0
## 119      Beverages         Dr Pepper (Small) 16.0      140 0.0     35
## 122      Beverages         Dr Pepper (Child) 12.0      100 0.0     26
## 123      Beverages    Diet Dr Pepper (Small) 16.0        0 0.0      0
## 124      Beverages   Diet Dr Pepper (Medium) 21.0        0 0.0      0
## 125      Beverages    Diet Dr Pepper (Large) 30.0        0 0.0      0
## 126      Beverages    Diet Dr Pepper (Child) 12.0        0 0.0      0
## 127      Beverages            Sprite (Small) 16.0      140 0.0     37
## 130      Beverages            Sprite (Child) 12.0      100 0.0     27
## 131      Beverages       1% Low Fat Milk Jug  1.0      100 2.5     12
## 132      Beverages Fat Free Chocolate Milk   1.0      130 0.0     22
## 133      Beverages Minute Maid 100% Apple J  6.0       80 0.0     19
## 134      Beverages Minute Maid Orange Juice 12.0      150 0.0     30
## 137      Beverages       Dasani Water Bottle 16.9        0 0.0      0
## 138    Coffee & Tea         Iced Tea (Small) 16.0        0 0.0      0
## 139    Coffee & Tea        Iced Tea (Medium) 21.0        0 0.0      0
## 140    Coffee & Tea         Iced Tea (Large) 30.0        0 0.0      0
## 141    Coffee & Tea         Iced Tea (Child) 12.0        0 0.0      0
## 142    Coffee & Tea        Sweet Tea (Small) 16.0      150 0.0     36
## 145    Coffee & Tea        Sweet Tea (Child) 12.0      110 0.0     27
## 146    Coffee & Tea           Coffee (Small) 12.0        0 0.0      0
## 147    Coffee & Tea          Coffee (Medium) 16.0        0 0.0      0
## 148    Coffee & Tea           Coffee (Large) 16.0        0 0.0      0
## 164    Coffee & Tea     Nonfat Latte (Small) 12.0      100 0.0     13
## 165    Coffee & Tea    Nonfat Latte (Medium) 16.0      130 0.0     16
## 176    Coffee & Tea Nonfat Latte with Sugar  12.0      140 0.0     13
## 197    Coffee & Tea Regular Iced Coffee (Sma 16.0      140 4.5     22
## 200    Coffee & Tea Caramel Iced Coffee (Sma 16.0      130 4.5     21
## 203    Coffee & Tea Hazelnut Iced Coffee (Sm 16.0      130 4.5     20
## 206    Coffee & Tea French Vanilla Iced Coff 16.0      120 4.5     19
## 209    Coffee & Tea Iced Coffee with Sugar F 16.0       80 4.5      1
## 210    Coffee & Tea Iced Coffee with Sugar F 22.0      120 7.0      2
```

Let's break this down. First, we have the `menu[]` template like we've been doing before to display portions of the `menu` variable. Next, if you look carefully, we have some stuff, followed by a comma, and then nothing: `menu[...,]`. This is exactly like what we did before where we extracted an entire row and displayed all columns. Only this time, instead of a row number, we're giving a true/false statement. We're referring to just the `Calories` column in the data frame, like we did before. Since each column in a data frame is a vector, we can treat it like we did with `fibs` above and find which of the elements meet the qualification. So, we're finding all rows such that the `Calories` column of that row is less than or equal to 150.

That's still a decent number of menu items, but if you look closely, they're mostly diet versions of sodas. We can apply another filter to the data and remove anything where the `Category` is `"Beverages"`. To do that, all we need to do is put an ampersand `&` after the filter but before the comma, and just type another filter.

```
menu[menu$Calories <= 150 & menu$Category != "Beverages",]
```

```
##            Category                     Item   Oz Calories Fat Sugars
## 39        Breakfast               Hash Brown  2.0      150 9.0      0
## 85           Salads Premium Bacon Ranch Sala  7.9      140 7.0      4
## 88           Salads Premium Southwest Salad   8.1      140 4.5      6
## 100 Snacks & Sides        Kids French Fries  1.3      110 5.0      0
## 101 Snacks & Sides               Side Salad  3.1       20 0.0      2
## 102 Snacks & Sides             Apple Slices  1.2       15 0.0      3
## 103 Snacks & Sides  Fruit 'n Yogurt Parfait  5.2      150 2.0     23
## 106        Desserts    Oatmeal Raisin Cookie  1.0      150 6.0     13
## 107        Desserts      Kids Ice Cream Cone  1.0       45 1.5      6
## 138   Coffee & Tea         Iced Tea (Small) 16.0        0 0.0      0
## 139   Coffee & Tea        Iced Tea (Medium) 21.0        0 0.0      0
## 140   Coffee & Tea         Iced Tea (Large) 30.0        0 0.0      0
## 141   Coffee & Tea         Iced Tea (Child) 12.0        0 0.0      0
## 142   Coffee & Tea        Sweet Tea (Small) 16.0      150 0.0     36
## 145   Coffee & Tea        Sweet Tea (Child) 12.0      110 0.0     27
## 146   Coffee & Tea           Coffee (Small) 12.0        0 0.0      0
## 147   Coffee & Tea          Coffee (Medium) 16.0        0 0.0      0
## 148   Coffee & Tea           Coffee (Large) 16.0        0 0.0      0
## 164   Coffee & Tea      Nonfat Latte (Small) 12.0      100 0.0     13
## 165   Coffee & Tea     Nonfat Latte (Medium) 16.0      130 0.0     16
## 176   Coffee & Tea Nonfat Latte with Sugar  12.0      140 0.0     13
## 197   Coffee & Tea Regular Iced Coffee (Sma 16.0      140 4.5     22
## 200   Coffee & Tea Caramel Iced Coffee (Sma 16.0      130 4.5     21
## 203   Coffee & Tea Hazelnut Iced Coffee (Sm 16.0      130 4.5     20
## 206   Coffee & Tea French Vanilla Iced Coff 16.0      120 4.5     19
## 209   Coffee & Tea Iced Coffee with Sugar F 16.0       80 4.5      1
## 210   Coffee & Tea Iced Coffee with Sugar F 22.0      120 7.0      2
```

Looks like that got rid of the sodas and juices, but we probably want to get rid of the coffee and tea as well if we really just want to display foods. We can add a third filter just as we added the second filter.

```
menu[menu$Calories <= 150 &
         menu$Category != "Beverages" &
         menu$Category != "Coffee & Tea",]
```

```
##            Category                         Item  Oz Calories Fat Sugars
## 39        Breakfast                   Hash Brown 2.0      150 9.0      0
## 85           Salads Premium Bacon Ranch Sala 7.9      140 7.0      4
## 88           Salads Premium Southwest Salad  8.1      140 4.5      6
## 100 Snacks & Sides            Kids French Fries 1.3      110 5.0      0
## 101 Snacks & Sides                   Side Salad 3.1       20 0.0      2
## 102 Snacks & Sides                 Apple Slices 1.2       15 0.0      3
## 103 Snacks & Sides   Fruit 'n Yogurt Parfait 5.2      150 2.0     23
## 106        Desserts     Oatmeal Raisin Cookie 1.0      150 6.0     13
## 107        Desserts       Kids Ice Cream Cone 1.0       45 1.5      6
```

Note that I spread the command onto multiple lines. To me, it's easier to read. R is not the most English-like language, so the more you can make it easier to read, the better you'll be when you come back to this code tomorrow. In fact, you can start to add comments to your code, using the `#` symbol:

```
# Filter out all the unhealthy stuff.
menu[menu$Calories <= 150 &  # Anything with 150 calories or less
         menu$Category != "Beverages" &  # not including soda and juice
         menu$Category != "Coffee & Tea",] # not including coffee and tea
```

We can apply other filters using other techniques as well. Let's say your getting food for your uncle Bob who, let's just say he doesn't have the best eating habits. For example, if you wanted to see which menu items have bacon in them and what coke products they offer.

4.3.3 Tangent: Packages

To do this, we're going to have to install a *package*. R packages are bundles of code that other people have written and made available for others to download. These usually contain several additional functions that plain ol' R can't handle very easily. The task we need to do is to check whether the menu item contains a specific word. This is called *pattern matching*. The best way I know of to handle this kind of task is to use `str_detect` function that's in the `stringr` library.

By the way, `stringr` is one of several packages that together comprise what's called the `tidyverse` and is written by Hadley Wickham. On November 11, 2017, I'll be doing a workshop specifically on `tidyverse`.

To download `stringr`, we first have to install it. We can do this with `install.packages`.

```
install.packages("stringr")
```

That'll run some code in your R Console and will take a few seconds to install. You'll need the internet for this too since we're downloading stuff. Once it's done, the code is installed to your computer, but you need to make it explicitly available for R to use in this session. To do that, use the `library` function (with the package name *not* in quotes).

```r
library(stringr)
```

Now, we have at our disposal a whole bunch of new functions specifically geared towards working with strings. Pretty cool.

4.3.4 Back to the data

Now that we have `str_detect` function, we can now filter our `menu` data frame and just show the items that have bacon in them. What the `str_detect` function does is it takes two arguments: the vector you want to filter, and some text (called a *string* in computer-speak) that you want to search for. The function by itself produces a long list of `TRUE`s and `FALSE`es, just like we saw before with the logical operators.

```r
str_detect(menu$Item, "Bacon")

##   [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
##  [23] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
##  [67] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [221] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [232] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [243] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [254] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

So when we incorporate that into the subsetting, we can get the list we want.

```r
menu[str_detect(menu$Item, "Bacon"),]

##           Category                          Item   Oz Calories Fat Sugars
## 7        Breakfast Bacon, Egg & Cheese Bisc  5.3      460  26      3
## 8        Breakfast Bacon, Egg & Cheese Bisc  5.8      520  30      4
## 9        Breakfast Bacon, Egg & Cheese Bisc  5.4      410  20      3
## 10       Breakfast Bacon, Egg & Cheese Bisc  5.9      470  25      4
## 20       Breakfast Bacon, Egg & Cheese McGr  6.1      460  21     15
## 21      Breakfast Bacon, Egg & Cheese McGr  6.3      400  15      16
## 25      Breakfast Bacon, Egg & Cheese Bage  6.9      620  31       7
## 26      Breakfast Bacon, Egg & Cheese Bage  7.1      570  25       8
## 52    Beef & Pork   Bacon Clubhouse Burger  9.5      720  40      14
## 54    Beef & Pork           Bacon McDouble  5.7      440  22       7
## 64 Chicken & Fish Bacon Clubhouse Crispy C 10.0      750  38      16
## 65 Chicken & Fish Bacon Clubhouse Grilled   9.5      590  25      14
## 68 Chicken & Fish  Bacon Cheddar McChicken  6.0      480  24       6
## 69 Chicken & Fish Bacon Buffalo Ranch McCh  5.7      430  21       6
## 85         Salads Premium Bacon Ranch Sala  7.9      140   7       4
## 86         Salads Premium Bacon Ranch Sala  9.0      380  21       5
## 87         Salads Premium Bacon Ranch Sala  8.5      220   8       4
```

We can filter this further and get only the items that have bacon *or* chicken by supplying a list of strings rather than just one.

```
menu[str_detect(menu$Item, c("Bacon", "Chicken")),]
```

```
##          Category                     Item   Oz Calories Fat Sugars
## 7       Breakfast Bacon, Egg & Cheese Bisc  5.3      460  26       3
## 9       Breakfast Bacon, Egg & Cheese Bisc  5.4      410  20       3
## 18      Breakfast Southern Style Chicken B  5.5      470  24       4
## 21      Breakfast Bacon, Egg & Cheese McGr  6.3      400  15      16
## 25      Breakfast Bacon, Egg & Cheese Bage  6.9      620  31       7
## 58 Chicken & Fish Premium Crispy Chicken C  7.5      510  22      10
## 60 Chicken & Fish Premium Crispy Chicken C  8.8      670  33      11
## 62 Chicken & Fish Premium Crispy Chicken R  8.1      610  28      11
## 65 Chicken & Fish Bacon Clubhouse Grilled   9.5      590  25      14
## 68 Chicken & Fish  Bacon Cheddar McChicken  6.0      480  24       6
## 69 Chicken & Fish Bacon Buffalo Ranch McCh  5.7      430  21       6
## 70 Chicken & Fish  Buffalo Ranch McChicken  5.2      360  16       5
## 72 Chicken & Fish Premium McWrap Chicken & 10.7      480  19       6
## 74 Chicken & Fish Premium McWrap Chicken & 10.5      450  18       6
## 78 Chicken & Fish Premium McWrap Chicken S 10.3      380  10      12
## 80 Chicken & Fish Chicken McNuggets (6 pie  3.4      280  18       0
## 82 Chicken & Fish Chicken McNuggets (20 pi 11.4      940  59       0
## 85         Salads Premium Bacon Ranch Sala  7.9      140   7       4
## 87         Salads Premium Bacon Ranch Sala  8.5      220   8       4
```

You could go on forever with way to subset your data. Here, I've shown just a couple ways to do some pretty useful things.

5 Basic Visualizations

The purpose of this workshop is to get you started with R, so we don't have time to cover all the ins and outs of visualizations in R. For now, it might be good to get you started with a couple visualization and to introduce a package called `ggplot2`. This package is written by the same programmer who did `stringr` and `tidyverse`, Hadley Wickham, and is very popular among R users.

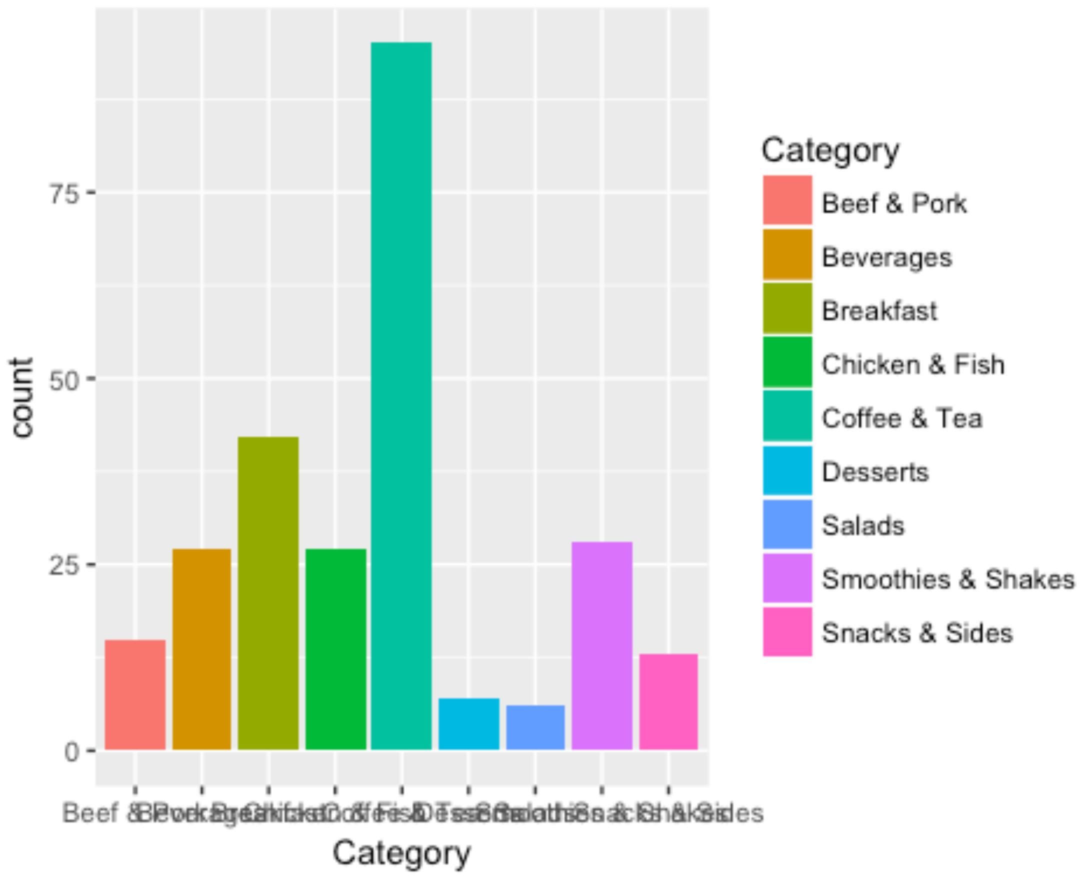
Let's get started by installing and loading `ggplot2`.

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

From here, we can run a series of commands to build a plot layer-by-layer.

First, let's just create a barplot that shows how many menu items in each category we have. To do this, we use the main `ggplot` function, where the first argument is the data frame we want to plot. The second argument in `ggplot` is `mapping` which takes list of various aesthetics, all wrapped up in the `aes` function. In our bar plot, we want each category to be its own bar going across the x axis. For fun, let's fill in the bars with one color per category. We then close that function, put a plus sign, and then do the `geom_bar` function. Doing two functions separated by a plus allows us to build the plot layer by layer. Here's the final result.

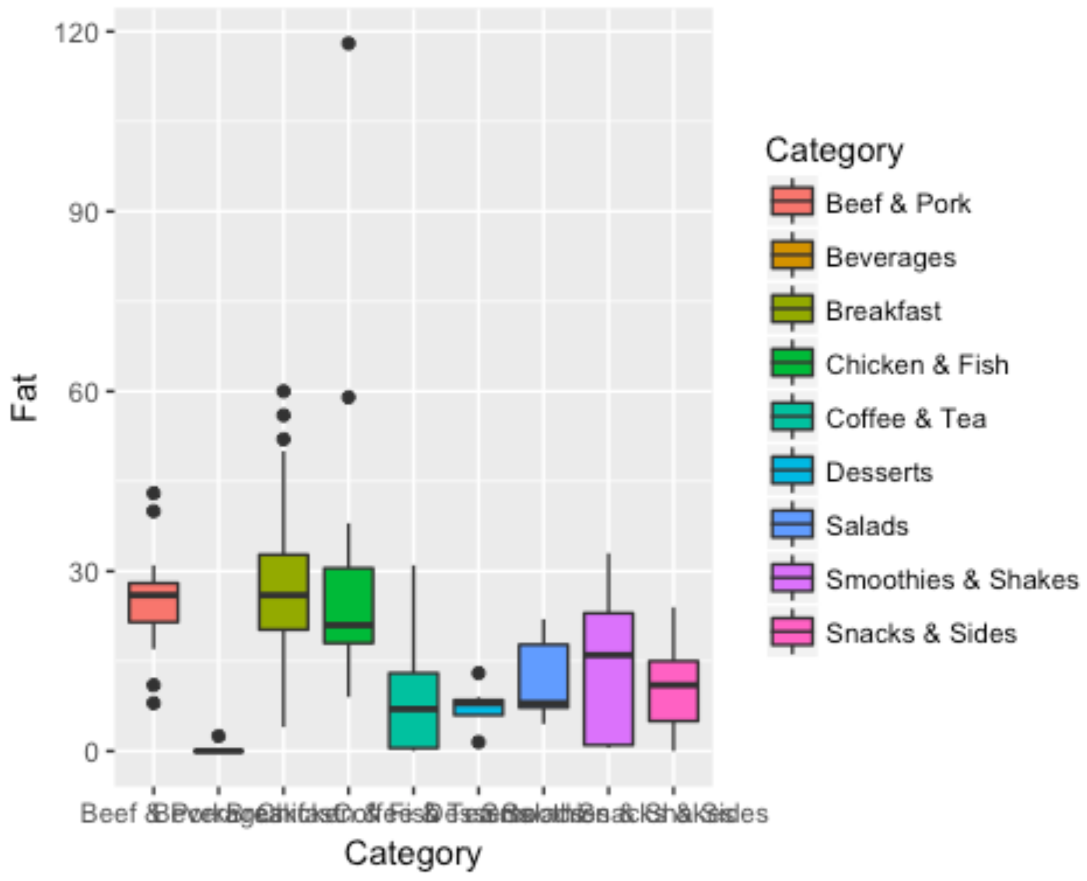
```
ggplot(data=menu, mapping=aes(x=Category, fill=Category)) +
    geom_bar()
```



Not bad! The biggest problem with this plot as is is that the names overlap along the bottom. We can change that as well as literally anything else on this plot (colors, order, x- and y-axis labels, the overall theme) later.

We start to do a more statistical approach and see the distribution of Fat per category. This is like what we had before but with two changes. First, we now need a *y* variable in the *aes* function. The *x* variable tells you what goes along the x-axis, and the *y* variable is what's along the y-axis, which in this case is *Fat*. We also use *geom_boxplot* instead of *geom_bar*.

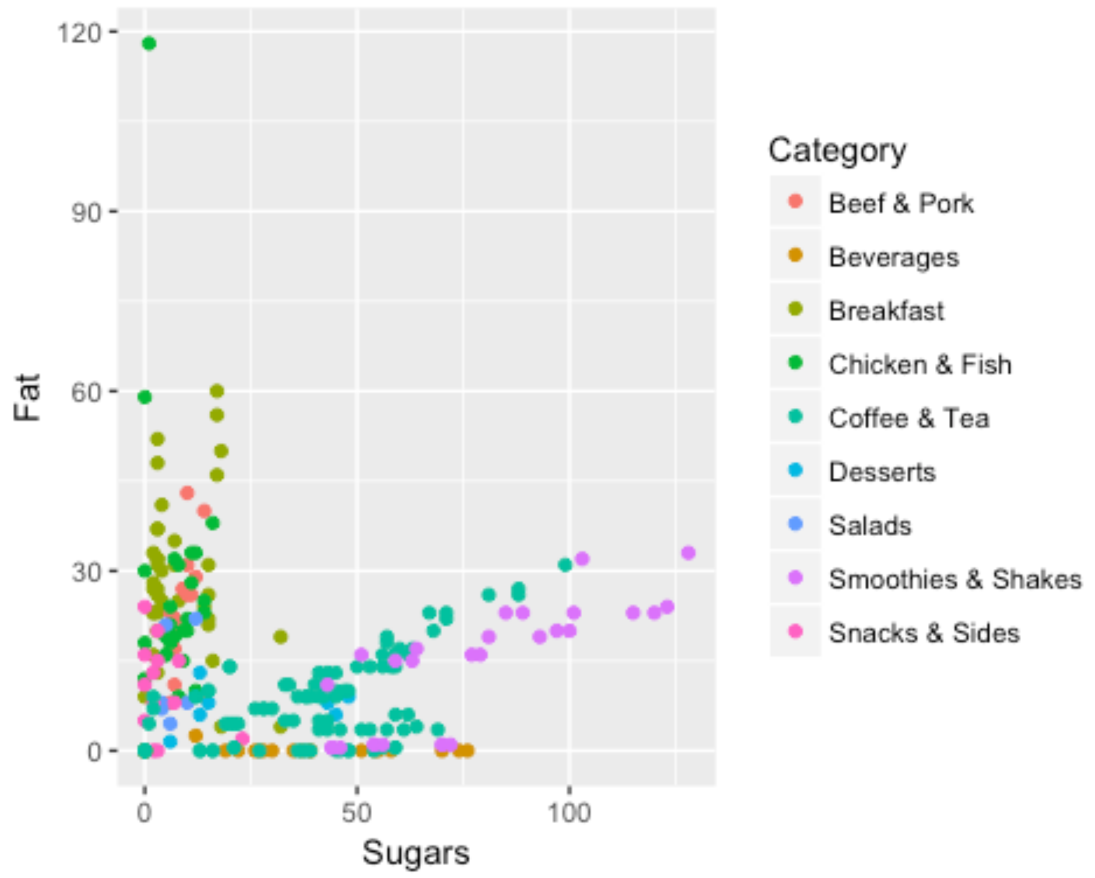
```r
ggplot(data=menu, mapping=aes(x=Category, y=Fat, fill=Category)) +
    geom_boxplot()
```



Here, we can see that breakfast items generally have the most fat, followed by Beef & Pork, and Chicken & Fish. Beverages have the least amount.

Finally, let's build a plot that shows the relationship between fat and sugar. Let's build a scatterplot. Along the x-axis, let's put *Sugars* and along the y-axis we'll do *Fat*. Here, instead of *fill=Category*, we'll put *color=Category* (*color* and *fill* are similar, but some plots look better with one instead of the other). Finally, we'll use *geom_point* this time to do a scatterplot.

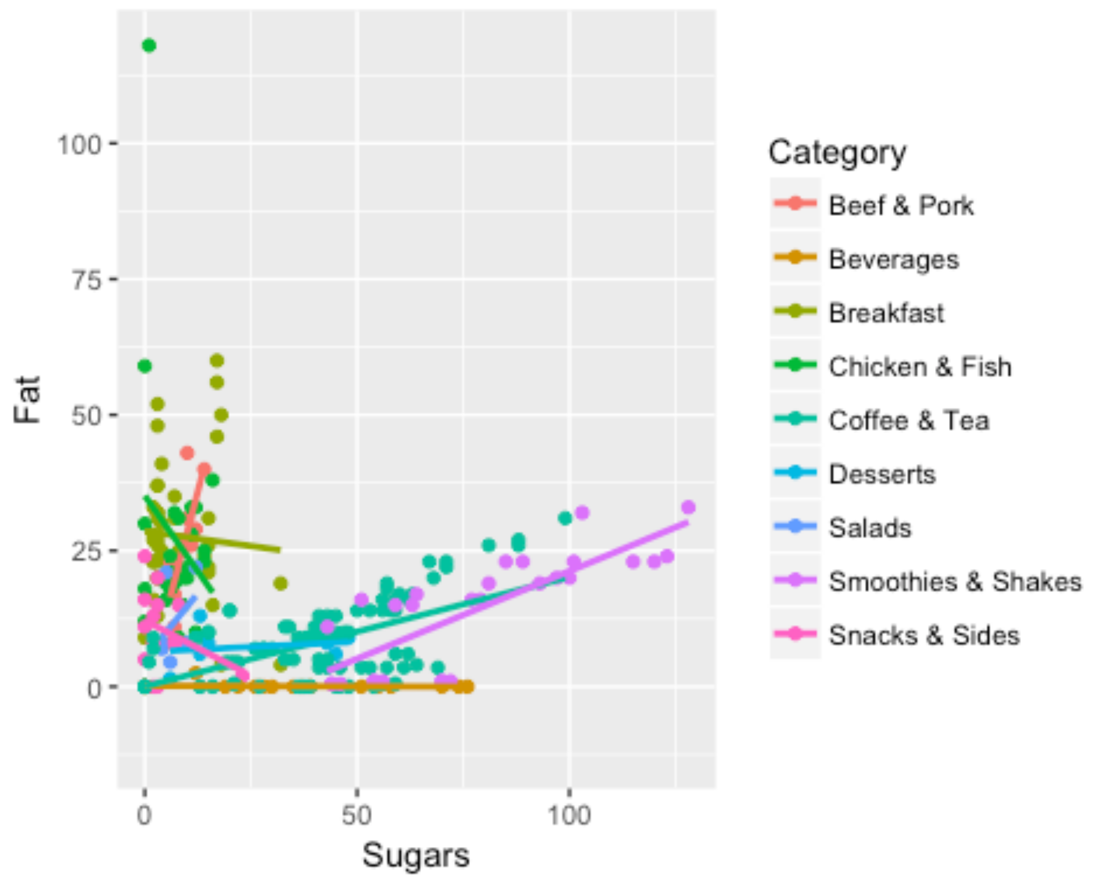
```r
ggplot(data=menu, mapping=aes(x=Sugars, y=Fat, color=Category)) +
    geom_point()
```



Here, we see that items generally have a lot of fat or a lot of sugar, but not both (there's nothing at the top right quadrant of the plot). There is some correlation between them though: as item have more fat they generally have more sugar, and vice versa.

In fact, we can even add a layer to this so we can add lines through the data to see the general trend per category. We can do this by adding *`geom_smooth(method="lm", fill=NA)`* to the function as a new layer.

```r
ggplot(data=menu, mapping=aes(x=Sugars, y=Fat, color=Category)) +
    geom_point() +
    geom_smooth(method="lm", fill=NA)
```



Now we can see that Smoothies & Shakes and Coffee & Tea have the most sugar generally and that the more they have, the more fat they have as well.

6 Where to go for help

6.1 In-R Help

R offers some good resources to help you learn about its functions. For example, if you put a question mark `?` before a function name, on the lower right quadrant of RStudio, some help will pop up.

```
?sqrt
```

These give documentation about how to use the function, what arguments it takes, and some example code. Usually, I scroll down to the example code and can find what I need there. It takes some experience to understand what the help pages do to be honest, but they eventually get to be very useful.

You can also precede a function name with two question marks `??` to search all of your installed packages for it.

```
??str_detect
```

This is useful if you know there's some sort of R function but you don't remember what package it's in. So if you get some code with the function `str_detect` in it but you forgot to load the `stringr` package, you can search for it using this double question mark and it'll tell you which ones to load.

6.2 Books and websites

Because R is so widely used, there are tons of resources out there. Here, I list just a few sites and books that I have found useful on a variety of topics.

- An Introduction to R by Venables, Smith, and the R Core Team (2017). This is a 99-page PDF that introduces R and some basic skills on how to use it. This version is only a few months old and it a thorough resource for beginners.
- This R Cookbook site is great and has helped me a lot.
- The *tidyverse* website is the launchpad for learning to use the `tidyverse` package.
- The publishing company Springer has a series called *Use R!* that has over 50 volumes in it. Many of them cover general skills like `ggplot2` or Shiny, but others are geared towards specific field like business, ecology, biostatistics, and general statistical procedures. The ones that might be most useful for beginners include *A Beginner's Guide to R* and *R Through Excel*. These books are all available as free PDFs to download through UGA's library

There are a lot of resources just on `ggplot2` specifically:

- The R Graphics Cookbook, and its accompanying *website* has helped me a ton for learning how to use `ggplot2`. It has great, clear examples on how to do stuff.
- Hadley Wickham, the creator of `ggplot2`, has some presentations available online *here* and *here*.
- The *ggplot2* website is is a good launching pad for other places to find help.
- The book *ggplot2:* Elegant Graphics for Data Analysis by Wickham is available for purchase, or, if you're a UGA student, available as a free download through the library's website.
- *Lynda.com*, which is free for UGA students, has some great help for learning R.

Your first stop really should just be google though, which will often take you to StackOverflow, YouTube, or other websites.

7 Conclusions

The goal for this workshop was to expose you to the kinds of things that are possible with R, to give you some exposure to a few R concepts, and to point you in the right direction to learn more. As a review, we covered the following ideas:

- What R is and how it compares to some alternatives
- The difference between R and RStudio.
- Basic R skill: Installing R and RStudio, variables, functions, getting data into R
- Displaying your data, extracting portions of it, and filtering, with tangents that covered logical operators and packages.
- Basic visualizations which introduced `ggplot2`.
- Where to find help.

This workshop is not enough to be a game-changer for you: you'll have to take the initiative to learn more on your own. But hopefully it has got you curious enough to make you want to learn more about R.