An Introduction to ggplot2

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

Presented at the UGA Willson Center DigiLab
Thursday, September 12, 2017

This is the second installment of the R workshop series. This document will cover some of the basics of data visualization in ggplot2: (1) some general principles of data visualization; (2) visualizing two continuous variables as scatterplots including adding additional variables to the plot; (3) plotting one variable with bar plots and histograms while demonstrating how to set general properties of the plot; (4) visualizing one continuous variable and one categorical variable with boxplots and violin plots with tangents on adding additional layers to the plot and consolidating code; (5) other ggplot2 functions including how to add labels, titles, and captions, the various built-in themes, and splitting the plot up into facets; (6) where to go for help, both in R and on the internet.

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

(Updated October 11, 2017)

1 Introduction

This workshop introduces the R package ggplot2. After some introductory discussions of visualizations and some basic data types, we dive right into how to make plots, improving code, and how to customize the look. This workshop does not teach every aspect of ggplot2, but instead exposes you to some basic code to make some basic plots, with the hopes that you leave being able to apply this code to your own data. To participate in this workshop, it is expected that you have some experience with R. I don't expect you to be a pro, but I'm assuming you have been able to get your data in and out of R, you've run some functions, and that you're familiar with the basics.

1.1 Data Visualization

What is the purpose of data visualization? We see charts, figures, graphs, and plots all over the place, but have you ever stopped to think about what it is that those are doing? It seems like we visualize data because we need to consume a lot of information all at once.

The data that we need to visualize most often takes the form of a table or spreadsheet of some sort. But unless it's very small, it's hard to get a good idea of trends and patterns. Some statistical methods are designed to summarize your data in various ways, including things like the mean, median, and standard deviation. But sometimes it's just nice to be able to "see" hundreds or thousands of raw data points all at once, without any summary statistics.

Data visualization also takes two forms, split up by the intended audience: the researcher and everybody else.

1. *For yourself*: Sometimes, all you need to do is create a quick-and-dirty graph so that you can get an idea of what's going on. These types of visualizations should be easy, quick, and informative. Little details like the aesthetics of the overall image are less important. Some kinds of data visualizations are meant specifically for the researcher and are not exactly intended to be included in any sort of publication. For example, plotting the residuals of a regression model using a Q-Q plot lets the researcher know the residuals are homoscedasatic. Looking at a scree plot helps find how many principal components to use in a PCA. These plots are important because they allow the researcher to gather information that can be used to determine future analysis, but often don't get seen by others.

2. *For others*: The other type of plot are those that are for public consumption. These are the ones you actually see on a webiste, presentation, or on the page. These are designed to convey specific information about some data in order to support some view.

The key to a good visualization is that it lets the data speak for itself. The addition of extra fluff (shadows, 3D, extravagnet colors) eclipses what the graph is actually showing. A good visualization is minimal. It is also faithful to the data, and doesn't misrepresent it by modifying axes or colors the wrong way.

Data visualization is as much of an art as it is a science. Yes it takes computational power to turn a spreadsheet into some beautiful graphic, but it takes an actual human to design the visual and make it appropriate with what you're showing. It should be thought of as an additional tool in order to help your audience understand the idea you're trying to convey.

1.2 Software

If you're anything like me, you've often felt like the ability to make professional charts and graphs has been impossible without some serious photoshopping skills. I've seen some really compelling visualizations of data in conferences and papers that really make it easy to summarize a lot of data in a single graphic. It sometimes isn't even anything fancy: a simple bar chart or scatterplot can add a nice visual touch to an otherwise text-heavy project.

The problem with making visualizations is that a lot of the existing software has some limitations. In my opinion, good data visualization software should have these properties:

1. *Customization*: I know of a website where I can upload my data and it will produce single stunning graphic. It's beautiful, but that's all it does. If I want to modify the colors, rotate it, add labels, change the font, or other make any other changes, I can't. Ideally, I should be able to customize whatever I want in my plot. And I mean *everything*. Font, line width, slight shades in colors, layout. Most software won't give you that.

2. *Professional*: Some software has the flexibility of making custom plots, but they all end up looking a bit cheesy. Yes, the information is conveyed, but it ends up looking like a middle-school science fair. You have no control over graphics like 3D shapes, shadows, and other details. Ideally, data visualization should produce stunning graphics that I would not be ashamed of showing at a conference presentation or using in a journal publication.

3. *Avoids carpal tunnel*: There is a lot of software out there that produces great graphics and you can customize it however you want, but it's extremely tedious. There just seem to be lots of clicks and menus you have to navigate through to get small changes. Sometimes changes have to occur in a specific order, and if you want to undo something, you have to do a lot of clicking again, or start all over. I don't like clicks. I think using the keyboard is easier on my hands and wrists, so writing code is preferable to me than any sort of menu or click-based software.

4. *Reproducibility*: Related the clicks is the idea that plots should be reproducible. Some software will let you add whatever you want to the plot, but you have to manually place things. This flexibility is desirable by some, but can be a pain for most people. For example, simple things like centering a title has to be eyeballed. Another problem with manual layout is that you'll never quite get the same plot twice. This is especially problematic if you want to create several similar plots that match because odds are they'll differ is slight (but frustratingly noticeable) ways.

Excel is a temporary solution, but you should not be satisfied with those plots. The direct link to your data is nice, but all those plots look awful and are a pain to customize. Last year I gave a presentation on JMP and discussed the visualizations that are possible. Like Excel these are not very professional looking and are very tedious to customize. Above all, Excel and other software only create a certain set of visualizations. If you want to create something brand new or an interesting combination of plots, it'll be very hard to do so in Excel.

1.3 ggplot2

One solution that satisfies at least all my demands in a visualization software is ggplot2. ggplot2 is an elegant and versatile R package that creates beautiful visualizations of data. It's an add-on package, meaning it's just a bunch of extra functions that have been written up and made available online for you to download. Its author is Hadley Wickham, who really has a knack and writing really good and useful R packages.

ggplot2 makes it easy to customize whatever you want in a plot. Yes, it comes with defaults, so if you just want quick and dirty visualizations, you can make those plots with no problem. But literally every aspect of the plots can be modified. This is what makes the plots look professional. Even the default settings aren't bad, and I have seen them in professional settings. But with just a little effort, you can make really nice graphics. This is all made easier because ggplot2 done entirely in R, meaning it's all written as code. No carpal tunnel here. This code-based nature of it is also what makes the plots perfectly reproducible every time, so making similar plots with different data is a breeze.

The reason why ggplot2 is so good is because it approaches the creation of visualizations a little differently than you might expect. It uses what's called the "Grammar of Graphics", based on a book of that title by Leland Wilkinson (and is available as a free eBook download through UGA's library!). In fact, that's what the "gg" in ggplot2 stands for. I don't have the time or space to go into detail about what this is, but the basic idea is that plots are built layer by layer. The fact that all the components of a plot is separated out makes them easier to manipulate and control, if you want the flexibility. It is also good because it just sort of takes care of everything for you, making it easy to use.

1.4 Data Types

Before we get too carried away, I want to emphasize something: *not all visualizations are meant for all kinds of data*. What do I mean by this? Just as certain statistical procedures require specific types of data, certain visualizations need certain types of data.

I've talked to people who wanted to make a scatterplot but when I took a look at their data, I saw that it was nothing but text, which doesn't lend itself to being a scatterplot. Scatterplots require at least two columns in your table -- variables as I'll refer to them from now on -- to be number-like. I've tried to help other people make other kinds of plots because they're flashy, sexy, and are used in other papers, but the important part is that you absolutely need the right kind of data.

The main two data types that I'll be refering to in this workshop is categorical and continuous data. *Categorical* data is something that can be grouped into distinct categories. These categories have no meaningful order and are mutually exclusive. Sometimes the number of categories can be small (glasses/contacts/nothing), relatively large (nationality, state of residence), or nearly inifite (favorite color, unique words). Some visualizations lend themselves well to categorical data, and some are better when there are fewer categories.

The other main kind of data is *numeric* or *continuous* data. These are numbers. These typically are things like measurements (height, weight, velocity, acoustic measurements, counting things, etc.) but can also be things like latitude and longitude. Sometimes it makes sense to have decimals (measurements, for example), and other times decimals don't make sense (counting things). There are lots of finer distinctions between subtypes of continous data, but for now we'll stick with just the basic concept.

1.5 Your Turn!

Think of your own data. What kinds of categorical variables do you have? What kinds of numeric data do you have?

2 The basics

In the last section we talked about what makes a good With the theoretical ideas out of the way, we're ready to start working in R.

2.1 Downloading and Installation

Because ggplot2 is an add-on package, you'll have to explicitly install it to your computer and then load it every time to run R. Luckily, this is pretty straightforward and can be done just like any other R package.

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

Alternatively, if you also use packages like `dplyr` or `tidyr`, you can load them all at once by installing and loading the `tidyverse` package, which includes all three (and more). I'll devote an entire workshop (if not more than one) on `tidyverse` next month.

2.2 Data for this workshop

The data that we'll be working with is a table of McDonald's menu items. This file contains some nutritional information such as calories, fat, and sugars, as well as the item name and category. It is available for free at Kaggle.com, where you can get complete nutritional information. You can read in this file directly from my website via R like this:

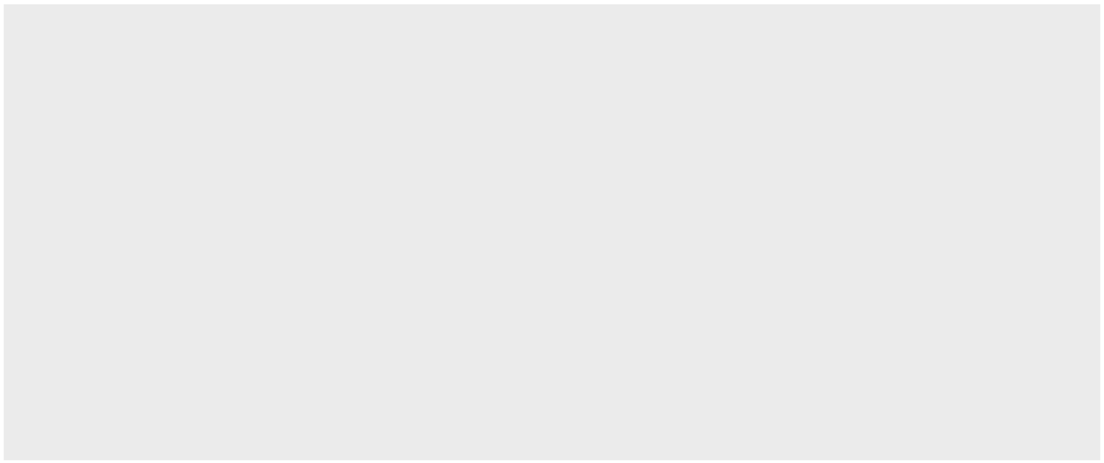
```r
menu <- read.csv("http://joeystanley.com/downloads/menu.csv")
```

When you use ggplot2 to make visualizations of your own data, you'll have to load it in and make sure it's clean and tidy just like the sample datasets are. I won't go over how to tidy your data, but a key part of creating good visualizations is good data.

2.3 Blank plots

Okay, finally, we're ready to plot! The main function in ggplot2 is the `ggplot` function. In fact, if you call this function without any arguments, it's still valid R code.

```
ggplot()
```



It doesn't do much, other than produce a gray rectangle though. In fact it's a coordinate system without any axes. This is the base layer that everything else gets added on top of. But it's important to see what your blank canvas is, so to speak.

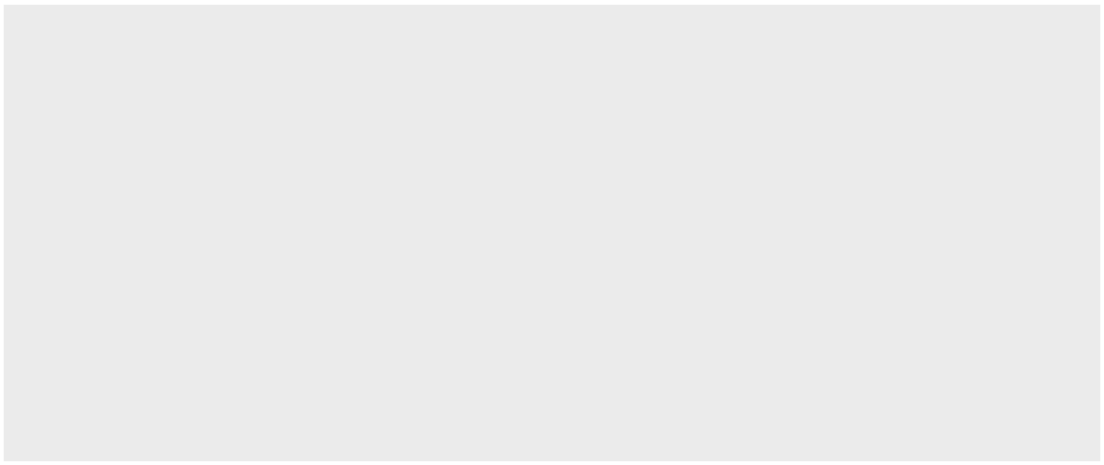
The first argument in the `ggplot` function is the `data` argument. To make a visualization of a particular dataset, just add `data=` plus the name of your dataset. We'll use the `menu` dataset that you should have downloaded earlier.

Let's inspect this dataset just so we have a better idea of what it looks like.

```
View(menu)
summary(menu)

##                 Category                                       Item
##  Coffee & Tea      :95   1% Low Fat Milk Jug                     :  1
##  Breakfast         :42   Apple Slices                            :  1
##  Smoothies & Shakes:28   Bacon Buffalo Ranch McChicken           :  1
##  Beverages         :27   Bacon Cheddar McChicken                 :  1
##  Chicken & Fish    :27   Bacon Clubhouse Burger                  :  1
##  Beef & Pork       :15   Bacon Clubhouse Crispy Chicken Sandwich:  1
##  (Other)           :26   (Other)                                 :254
##        Oz            Calories           Fat               Sugars
##  Min.   : 1.000   Min.   :   0.0   Min.   :  0.000   Min.   :  0.00
##  1st Qu.: 6.775   1st Qu.: 210.0   1st Qu.:  2.375   1st Qu.:  5.75
##  Median :12.000   Median : 340.0   Median : 11.000   Median : 17.50
##  Mean   :12.803   Mean   : 368.3   Mean   : 14.165   Mean   : 29.42
##  3rd Qu.:16.000   3rd Qu.: 500.0   3rd Qu.: 22.250   3rd Qu.: 48.00
##  Max.   :32.000   Max.   :1880.0   Max.   :118.000   Max.   :128.00
##
```

Okay, now that we have some idea of what we want to look at, let's now add this dataset to the `ggplot` function.

```
ggplot(data=menu)
```



Great. All this does is create that same blank gray rectangle. ggplot2 is smart but it's not *that* smart: you'll have to tell it what to do with the data. The way to add layers is typically through one or more `geoms`. The full list is long, but some of the functions that you might use include `geom_point()` for scatterplots, `geom_boxplot()` for boxplots, `geom_bar()` for bar charts, and `geom_map()` for maps. For the rest of the workshop we'll be working with several `geoms` one at a time and discussing how to use and modify them.

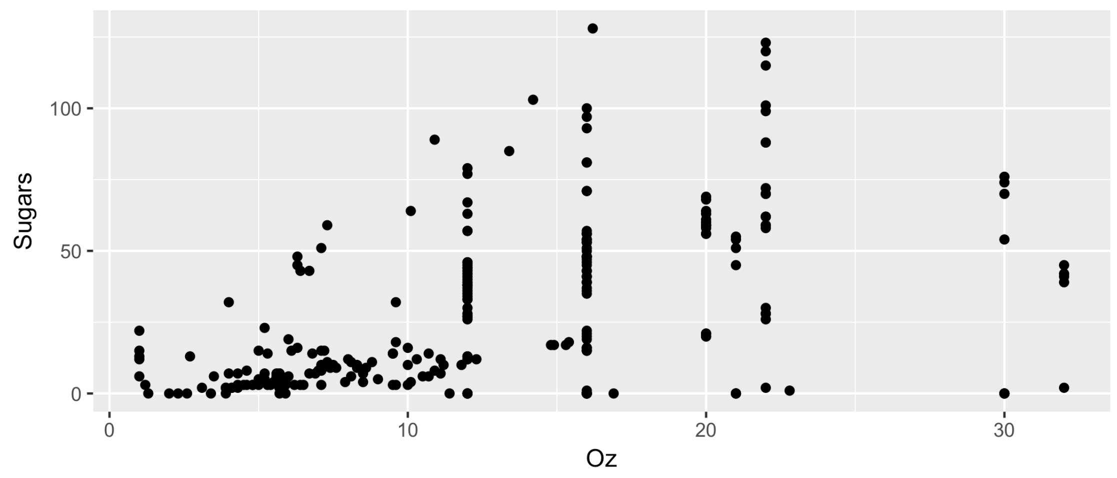
3 Two continuous variables

The most efficient way of showing two numberic variables at the same time is probaby going to be scatterplots. In this section we'll also look at how to add more variables to your plot with the addition of aesthetics like color, shape, and size.

3.1 `GEOM_POINT`

Unlike the `ggplot` function, you must provide some arguments to `geom_point()` (and all the other `geoms` for that matter) . You do this with the `mapping=` argument, which wraps up the various aesthetics of the plot inside the `aes()` function. Let's make a scatterplot of weight of the menu items and how much sugar each has. Keep in mind that a scatterplot typically requires two variables that are numbers, so weight (in ounces) and sugars (in grams) will work fine. We'll put weight in the *x*-axis and sugars in the *y*-axis, separated by a comma. Note that the names of these in R must match exactly how they look in the column headers (including capitalization).

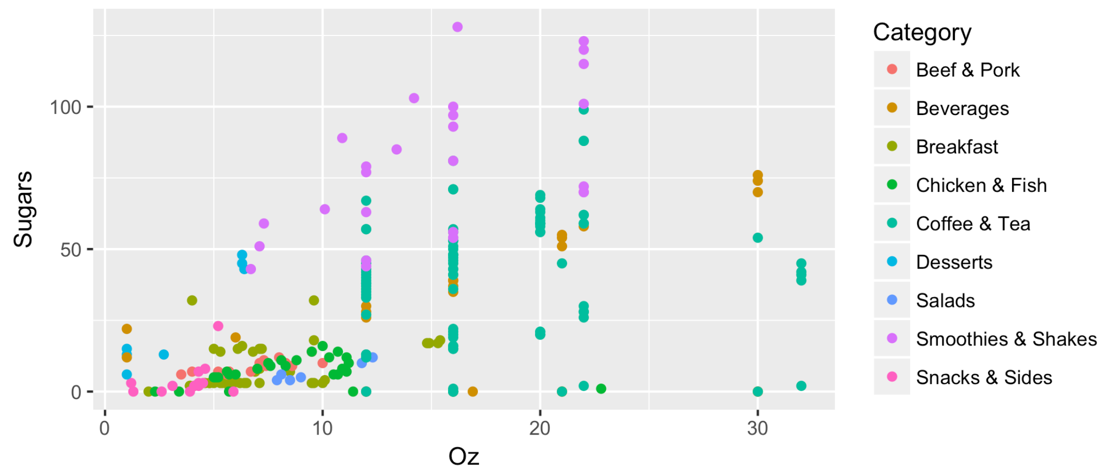
```
ggplot(data=menu) +
    geom_point(mapping = aes(x=Oz, y=Sugars))
```



Boom. We just created a visualization. With just two short lines of code, we created a decent scatterplot. Note that the gray background is still there and that we've overlaid white grid lines (major and minor ones if you look closely), *x*- and *y*-axis labels and tick marks, and of course the points themselves, which are black circles by default. You can change every one of these layers by the way and we'll get to some of those later.

We can modify this chart by adding more aesthetics. Sometimes it's useful to have the colors vary depending on some other variable. In this dataset, we have a column that refers to each category of menu item. To add that to the plot, we add it as a third aesthetic:

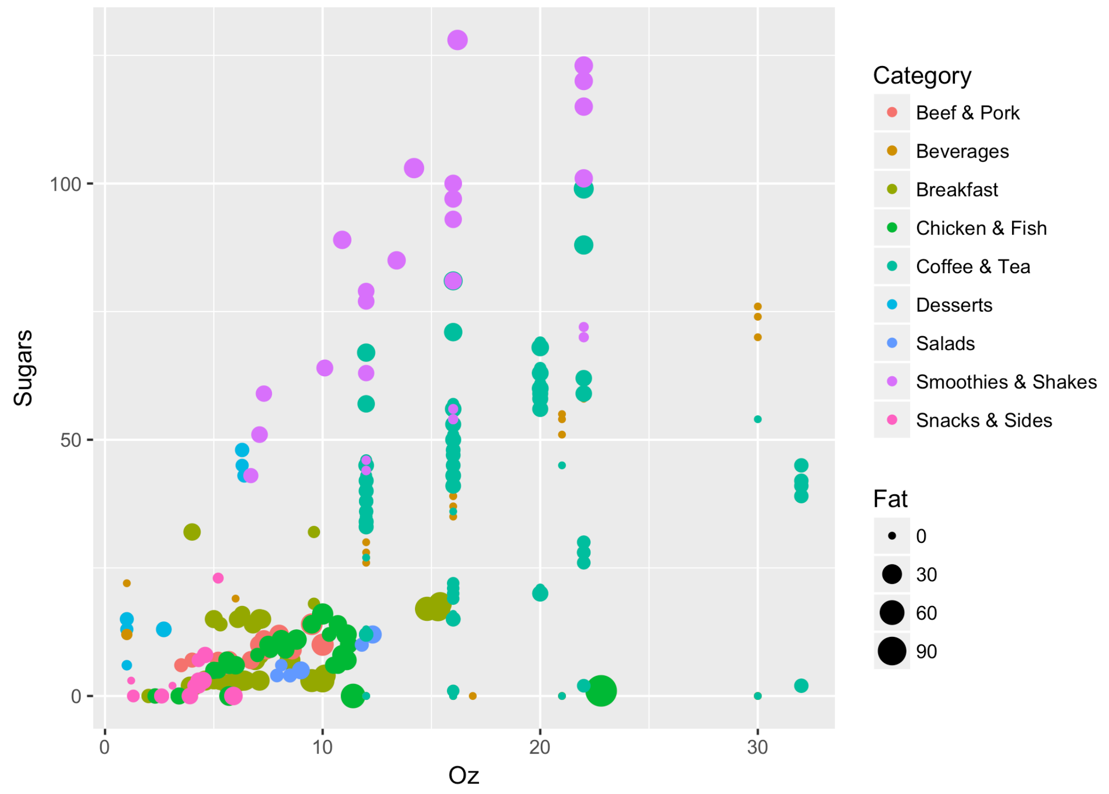
```
ggplot(data=menu) +
    geom_point(mapping = aes(x=Oz, y=Sugars, color=Category))
```



What `ggplot` has automatically done is determine what all the categories are in your dataset and assign them each a color. The categories are in alphabetical order by default and the colors are equidistant shades from red to purple. You can modify both the order and the specific colors, but for the purposes of this workshop we'll have to skip that.

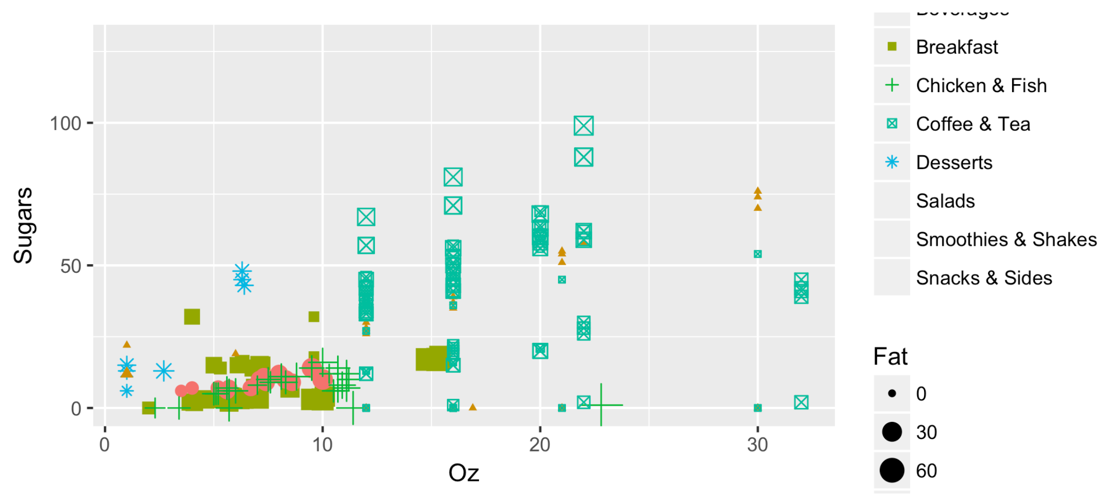
There are other aesthetics that you could add as well, either as additional elements to your plot or combined with what you have. For example, we could vary the size of the points depending on how much fat there is in each menu item.

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



If we wanted to really emphasize the differences in category, we could add shape as well.

```
ggplot(data=menu, fig.height = 5) +
    geom_point(mapping = aes(x=Oz, y=Sugars, color=Category, size=Fat, shape=
Category))

## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 9.
## Consider specifying shapes manually if you must have them.

## Warning: Removed 47 rows containing missing values (geom_point).
```



This is where things get crazy, and ggplot lets you know that with a warning message. First off, shapes are only good for categories where there are just a few options, like less than 6. So though it is theoretically possible to include all these aesthetics into your plot, that certainly doesn't mean you should. Right now we have four different pieces of information plotted on a single plot: Ounces is on the left-to-right dimension, Sugars is top-to-bottom, category is color and shape, and fat is size. That's a lot of things for a human to process. It's to the point that it's hard to draw any meaningful conclusions from the plot.

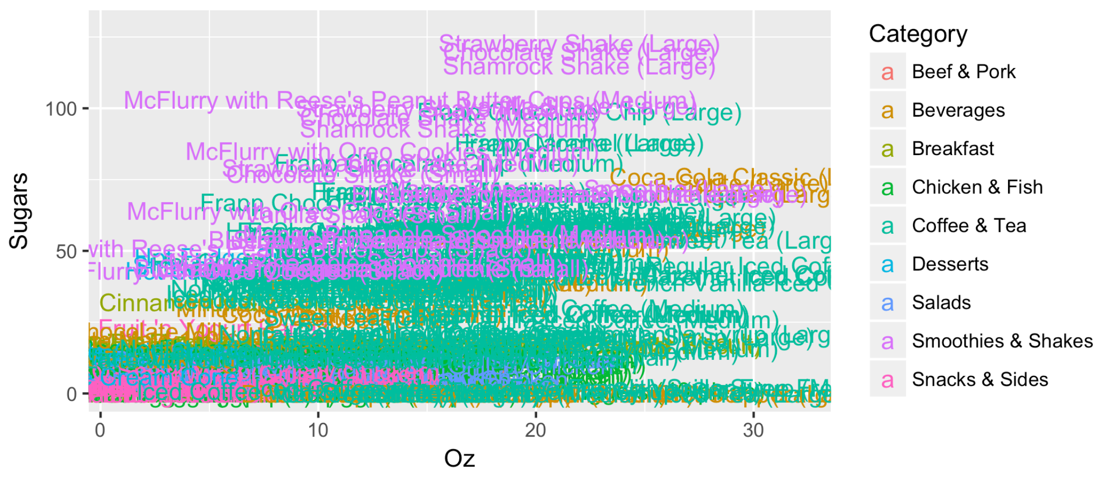
This gets into the idea of why we make visualizations in the first place. We want to display a large amount of data in a way that makes it easy to digest and see patterns. By throwing all these variables into one plot, we don't accomplish this purpose. It would be better to make separate plots that are easy to understand than one mega plot with everything.

3.2 GEOM_TEXT AND GEOM_LABEL

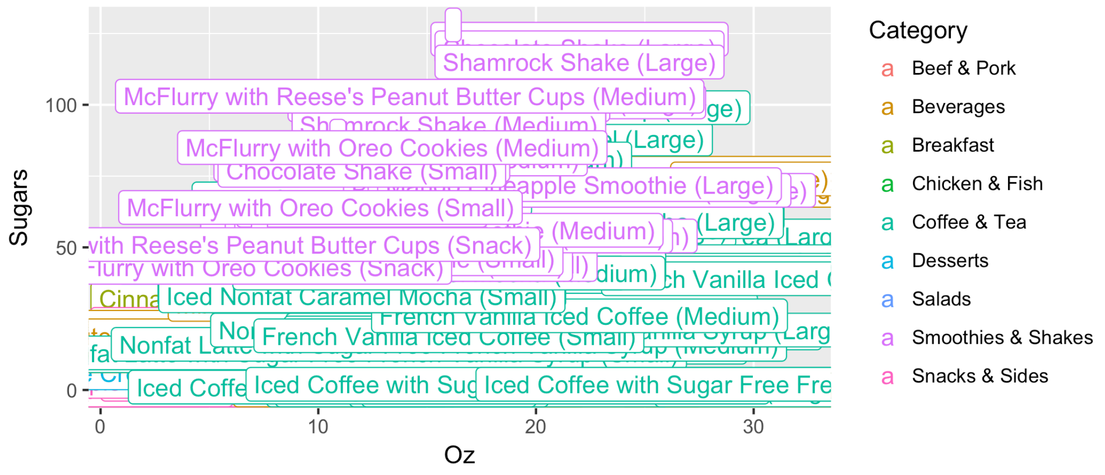
As an added bit of information, sometimes it's nice to see the name of the menu item instead of just a point. We can accomplish this either by using `geom_text` or `geom_label`. They are essentially the same thing, but the latter is a little easier to read.

To use either of these, we need to add one more aesthetic: `label`. The column you'll select is the one that'll be displayed on your graph instead of the dot.

```
ggplot(data=menu) +
    geom_text(mapping = aes(x=Oz, y=Sugars, color=Category, label=Item))
```



```r
ggplot(data=menu) +
    geom_label(mapping = aes(x=Oz, y=Sugars, color=Category, label=Item))
```



Because we have so many points and the names are so long, it's a bit of a mess. But this is a useful trick to be aware of if you have a smaller dataset to plot or if the labels are very short. For example, in linguistics, it's often nice to make scatterplots where each individual word is displayed rather than dots. In your dataset you might use names of states, cars, people, events, etc. which can make your graph a lot more useful.

3.3 Your Turn!

1. Continue to modify the scatterplot using this data. Try putting different variables in for the various aesthetics. Try removing some aesthetics and see how that changes the plot. Imagine you have something you want to convey to an audience about McDonald's menu items with this data; make the best plot you can that most clearly shows that idea.

2. What kinds of things can be shown using a scatterplot of your own data. Be specific: what aesthetics would you use and what information would you associate with each?

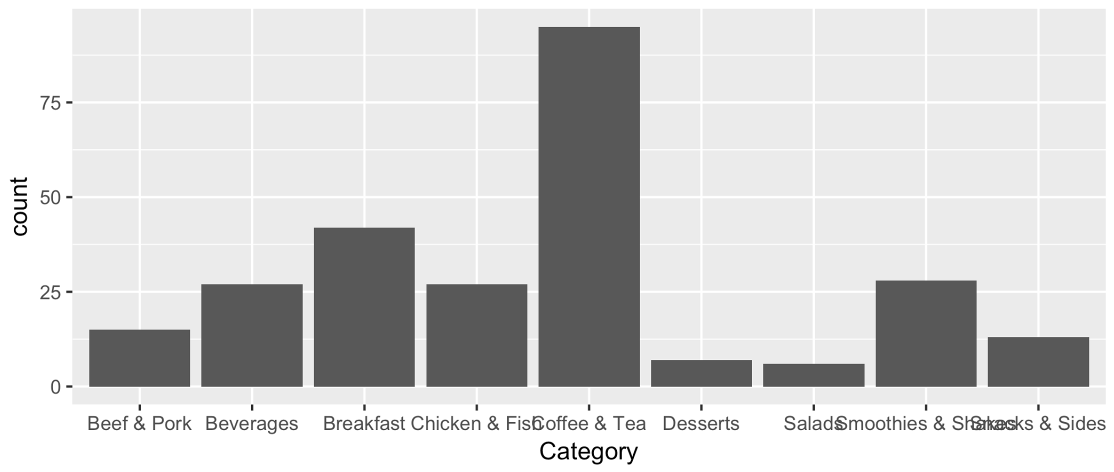
4 One variable

With scatterplots and the addition of colors, shapes, and sizes, it's easy to display a lot of variables all at once. Sometimes we just need to simplify things and just show one variable. One way to do this is with bars. Here we'll learn about these bar charts and their relative, the histogram. We'll also look at modifying global properties to your plot, rather than having them alternate with a variable.

4.1 geom_bar

Barplots are usually used when displaying how many of each category there are in a categorical variable. So here, we might want to use a bar plot to show how many menu items of each category there are. To do this, we can use the `geom_bar` function. The only aesthetic we need is just the `x` argument, which would be the column that contains the categories that you want each bar to belong to. In this dataset, the name of that column is `Category` coincidentally.

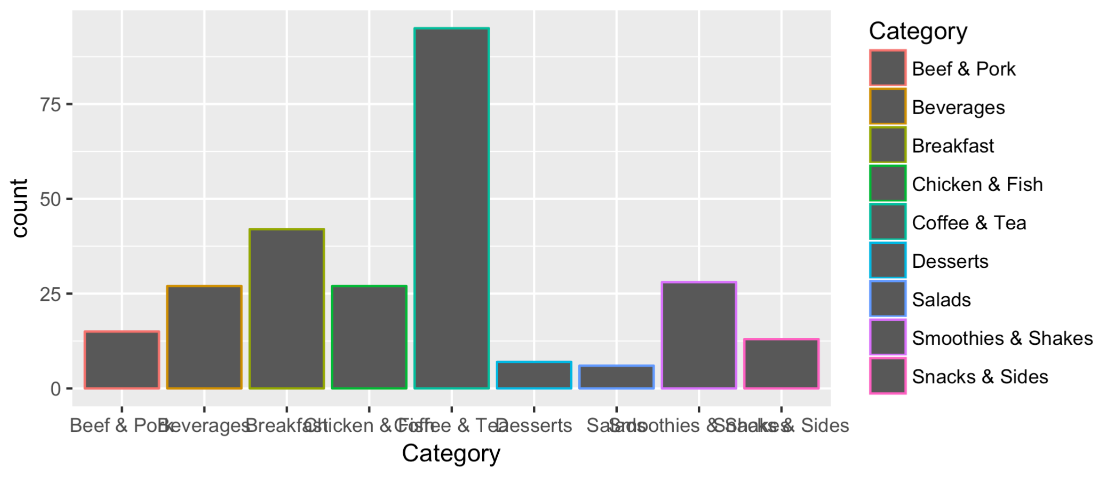
```
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category))
```



Notice what this plot does. Starting with the gray background layer, it superimposes several things: labels across the bottom for each menu category with an axis label (`Category`), tics across the left side with an axis label (`count`), another set of major and minor white grid lines, and the bars themselves. Here, we can see that the majority of menu items are actually coffee and tea products. This is likely because each individual size is treated as a separate item in this dataset because each size has its own nutritional amounts.

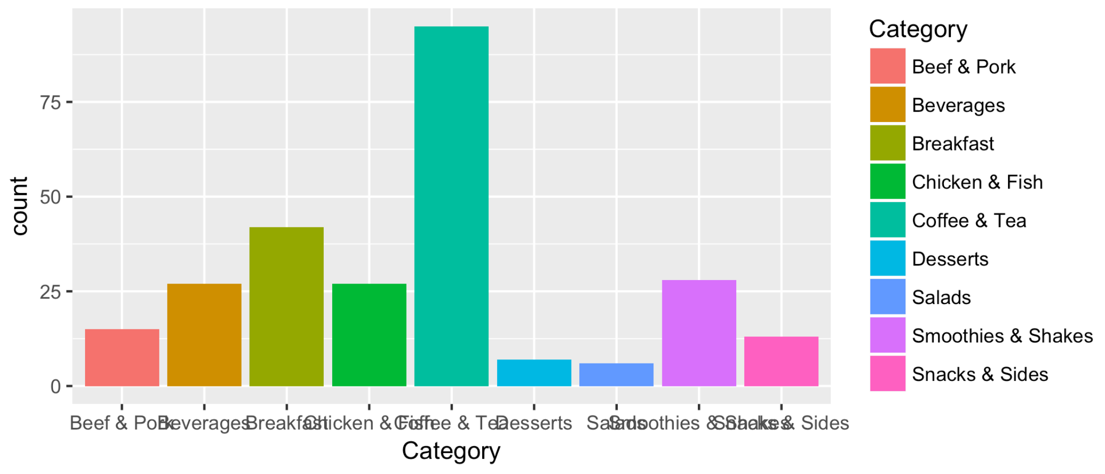
Right now, this is okay, but it's admittedly kind of boring. While we want simplicity in our plots, we do want them to look somewhat professional. We can change a few of the aesthetics to spruce it up a little bit. Just as with the scatterplot, we can add the `color` aesthetic, to give each category its own color.

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



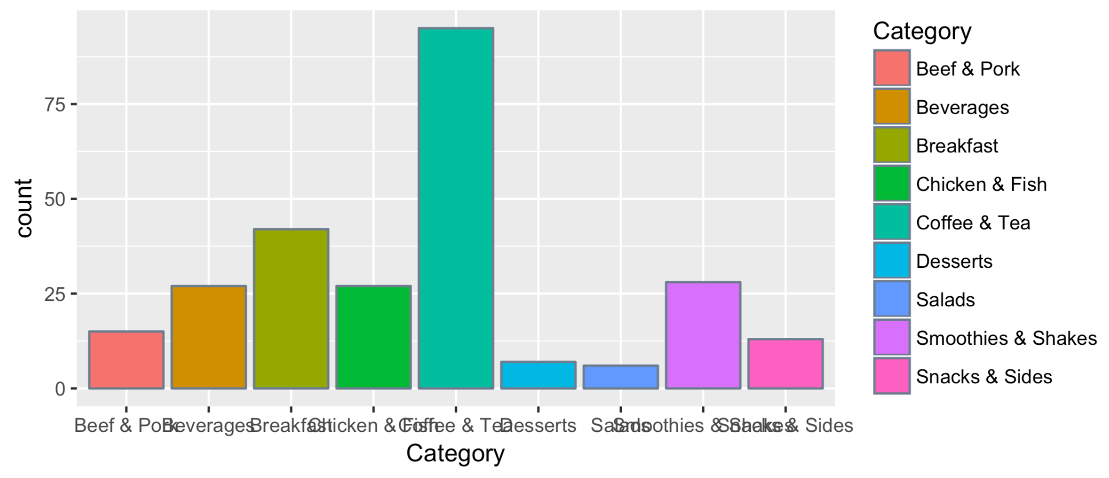
Oops! What did this do (other than create a kinda cool looking plot)? For `geom_bar`, the `color` aesthetic changes the outline. If we want to fill it in, we have to use `fill`:

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



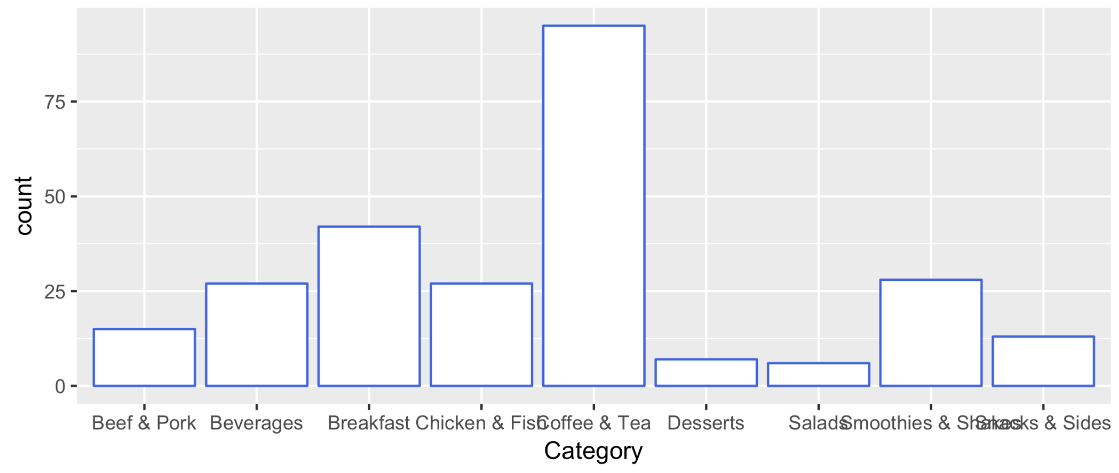
But what if we want all of the bars to be their own color but the outline to be just one color? If we want to apply some property to all items within a `geom`, we add that property *outside* of the `aes` function as a separate argument. For example, we can make the outlines all black while still retaining the colored bars. (Here, I use the color `slategrey`. You can find a complete list of colors in R by going here.)

```r
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category, fill=Category), color="slategrey")
```



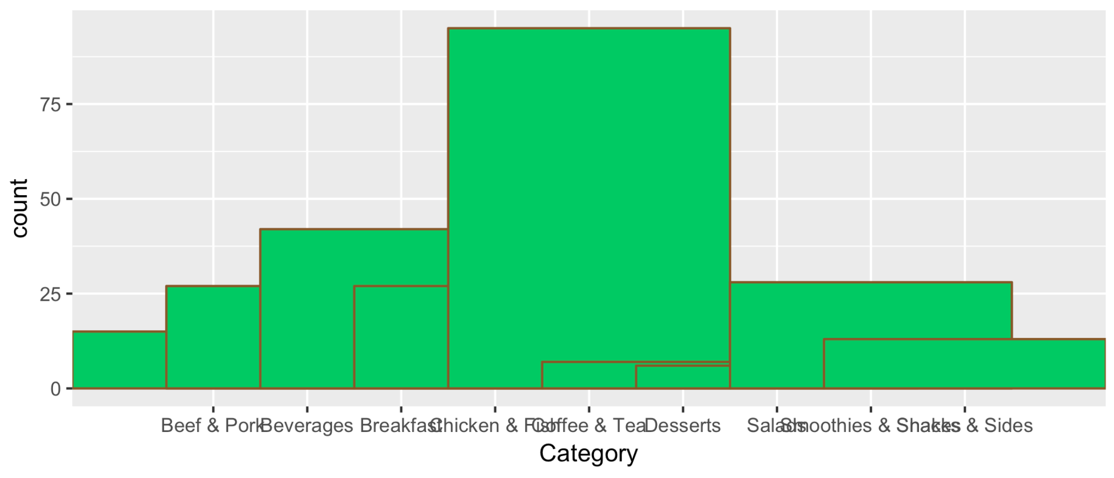
At this point, the sky is the limit as far as colors. You can set `fill` or `color` to be global properties or as aesthetics of the plot, or some combination of the two. For a minimalistic approach, you could set them both as global options and find an outline color that matches your powerpoint slides or something.

```
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category), fill="white", color="royalblue")
```

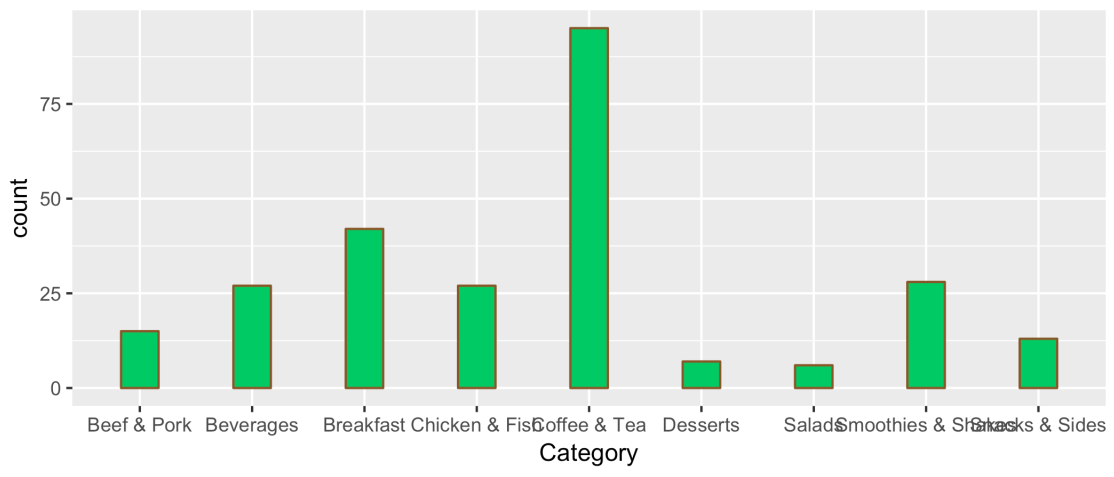


There are a couple other things you can adjust for this plot. If you want to tweak the width of the bars themselves, you can do so with the `width` property. The default is `1`, so use a larger number for wider bars and a smaller number for skinnier bars. Note that this is not an aesthetic, so it can't vary by some variable. In other words, it has to be an argument of `geom_bar()` rather than `aes()`.

```
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category), fill="springgreen3", color="tan4", wi
dth=3)

## Warning: position_stack requires non-overlapping x intervals
```

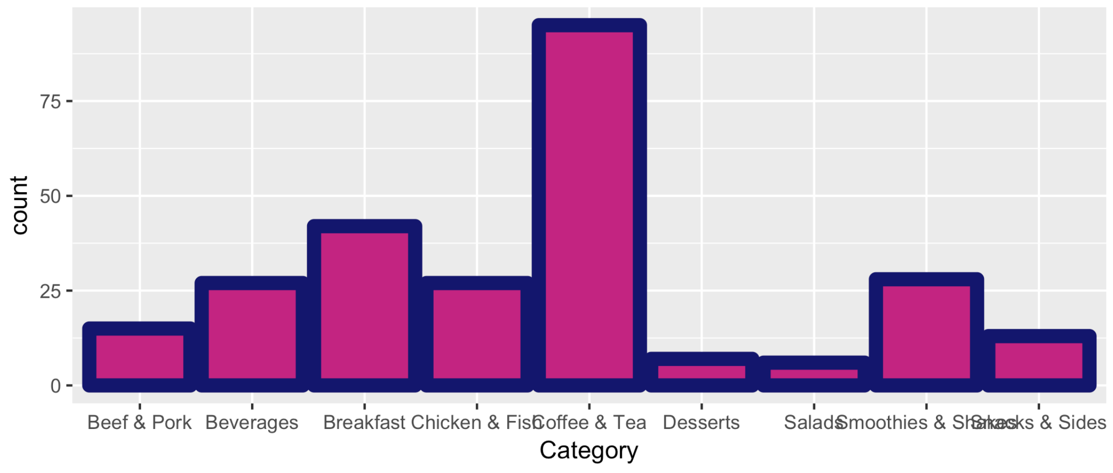


```r
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category), fill="springgreen3", color="tan4", wi
dth=1/3)
```

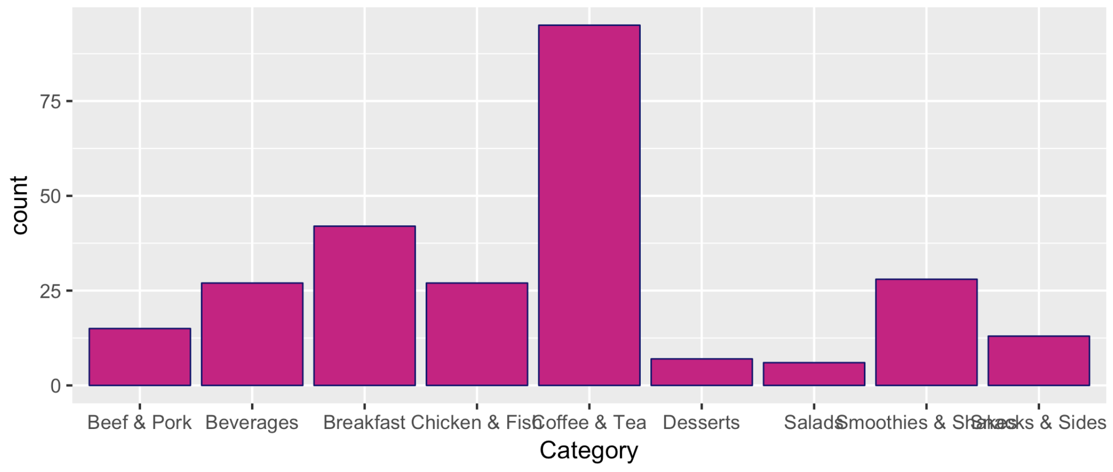


You can even change the thickness of the outline using the `size` property. Just like `width`, this must be a global property, and the default is `1`.

```r
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category), fill="mediumvioletred", color="midnig
htblue", size=3)
```



```
ggplot(data=menu) +
    geom_bar(mapping = aes(x=Category), fill="mediumvioletred", color="midnig
htblue", size=1/3)
```

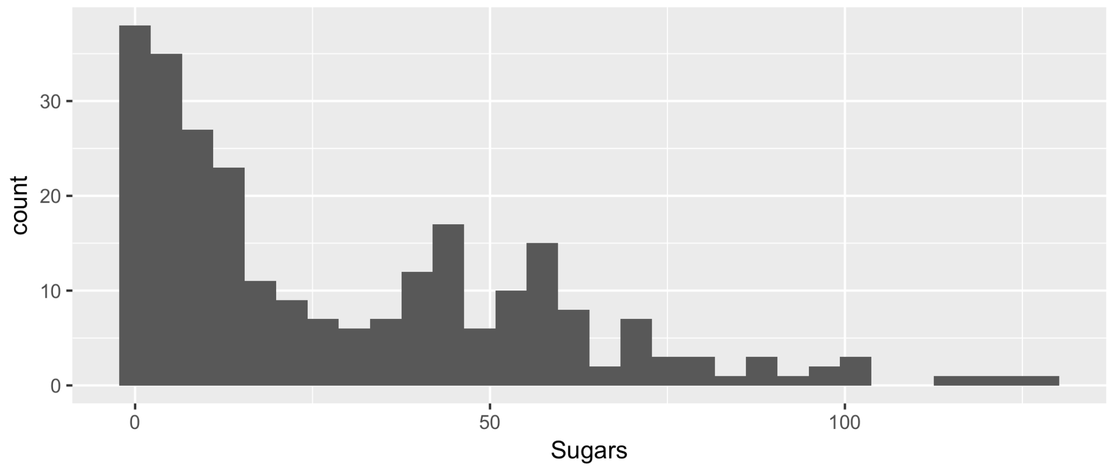


So barplots are good because you can quickly show how many of each category there are. I prefer them to pie charts because judging the height of bars is easier than the angle of pie wedges.

4.2 GEOM_HISTOGRAM

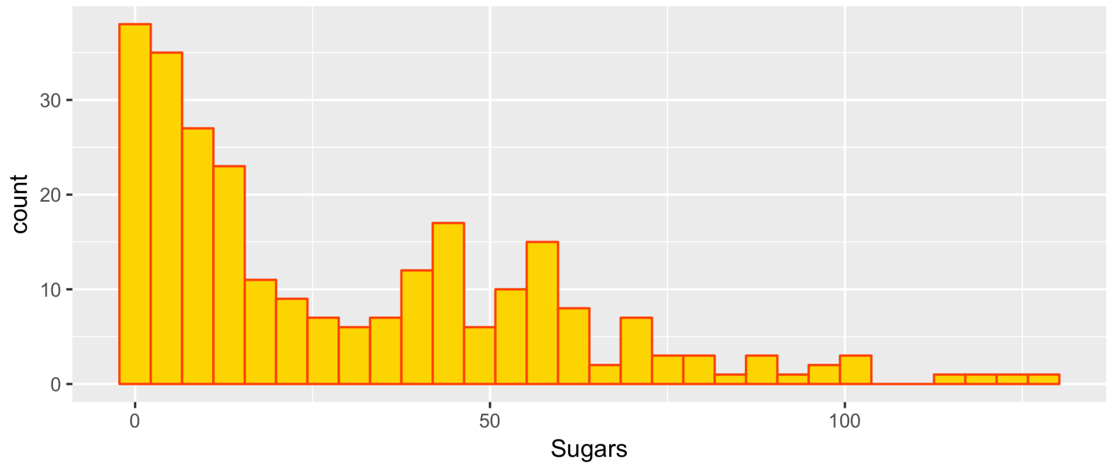
Related to the bar plot is the histogram. On the surface, the share a lot of similarities, but their underlying data is different. For the bar plot, we supplied `ggplot` with a categorical variable: with discrete, unordered categories. A histogram makes a similar plot with a numeric variable, so you can see the distribution of the data. Let's see the distribution of fat within a histogram.

```
ggplot(data=menu) +
    geom_histogram(mapping = aes(x=Sugars))

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



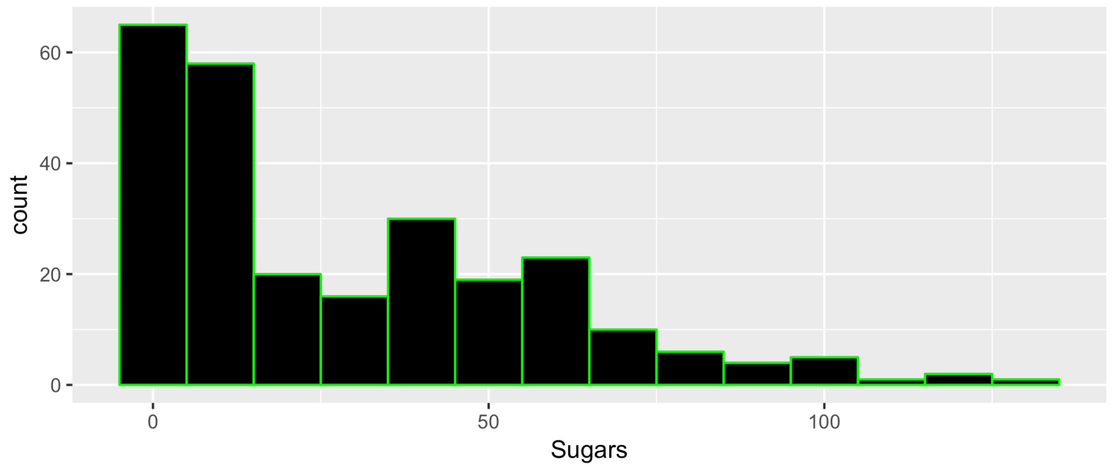
Here we can see that the majority of menu items are on the lower end of this graph. But this chart is a little hard to read since columns are all right next to each other. We can change this using the `fill` and `color` attributes just as before:

```
ggplot(data=menu) +
    geom_histogram(mapping = aes(x=Sugars), fill="gold", color="orangered")

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

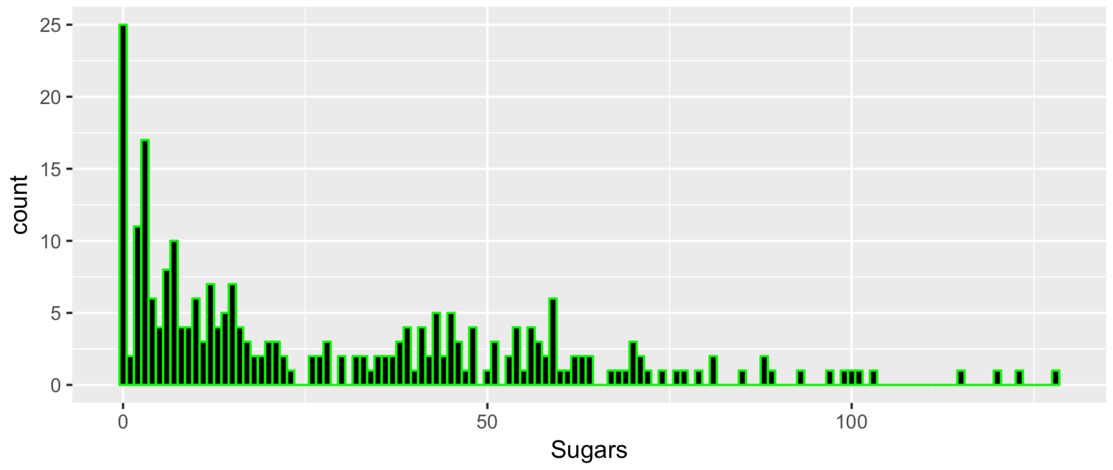


You can change the width of the bars, but this is a little bit different with `geom_historgram` than with `geom_bar`. Since we're dealing with numbers, a wider bar would take up more space on the x-axis, meaning it eats up more numbers. For example, the default width here is about 5 units per bar. So the furthest bar to the left shows how many menu items have 0--5 grams of fat. If we made that bar wider, it might cover 0-7 on the graph, therefore changing the amount of data, which changes the height of the bar. For histograms, the width of these bars is called the `binwidth` since we're dividing the data up into different "bins." If you change it to smaller or wider bins, notice how the shape of the graph changes.

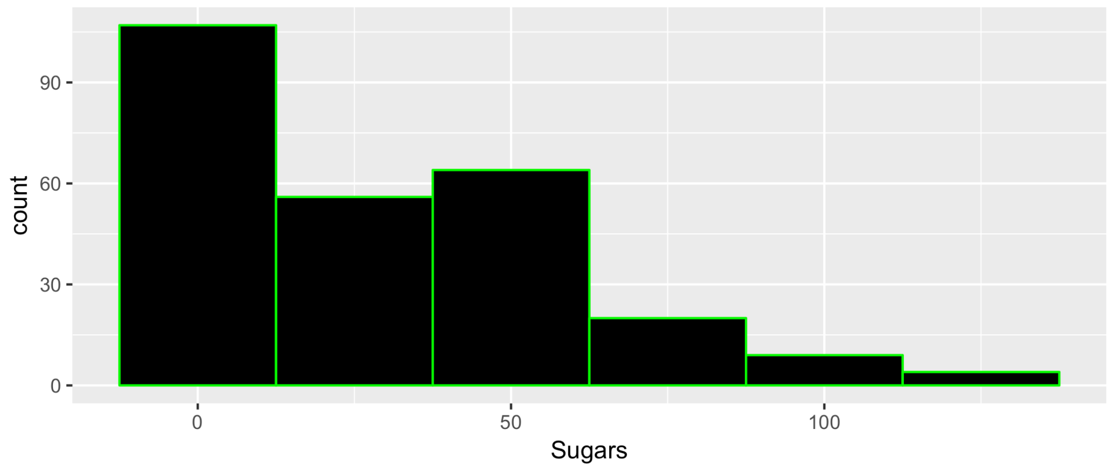
```
ggplot(data=menu) +
    geom_histogram(mapping = aes(x=Sugars), fill="black", color="green", binw
idth=10)
```



```
ggplot(data=menu) +
    geom_histogram(mapping = aes(x=Sugars), fill="black", color="green", binw
idth=1)
```



```
ggplot(data=menu) +
    geom_histogram(mapping = aes(x=Sugars), fill="black", color="green", binw
idth=25)
```



When creating a histogram, it's good to choose a good binwidth. There are algorithms made that determine what the best width should be, but it's probably easiest eyeball it, making sure that it's not too wide or narrow and that it's not hiding or emphasizing anything that isn't really there.

4.3 Your Turn!

1. Create two charts: one that shows how many items there are per category and one that shows the distribution of fat in these menu items. Make them look as professional as you can by altering the aesthetics.

2. Think of your own data: when might you use a chart like these. Would you use bar chart or a histogram? How do you know?

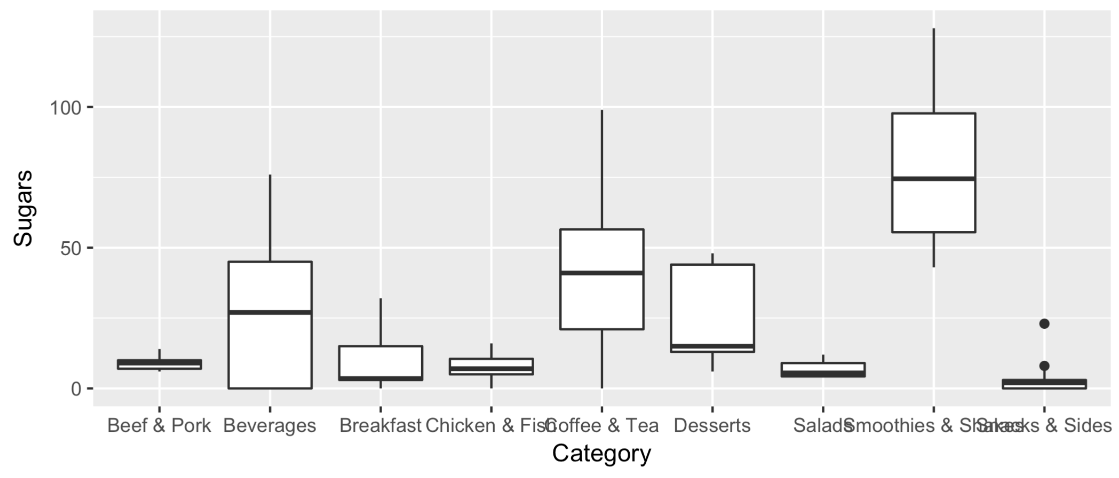
5 One continuous variable and one categorical variable

So far we've looked at what kinds of plots you can do with two continuous variables, and with just one variable (variable or continuous). The natural extension to this is what happens when you have one categorical variable and one continuous variable. We'll start with the more typical boxplots but then move on to violin plots. We'll also cover how to add even more layers to your graph.

5.1 `geom_boxplot`

A box(-and-whisker) plot is something you might learn about in statistics because it does a decent job at summarizing your data. It shows the average, distribution, and outliers of your data. We can make a basic boxplot using the `geom_boxplot` function. Here, we need a categorical `x` variable and a numeric `y` variable.

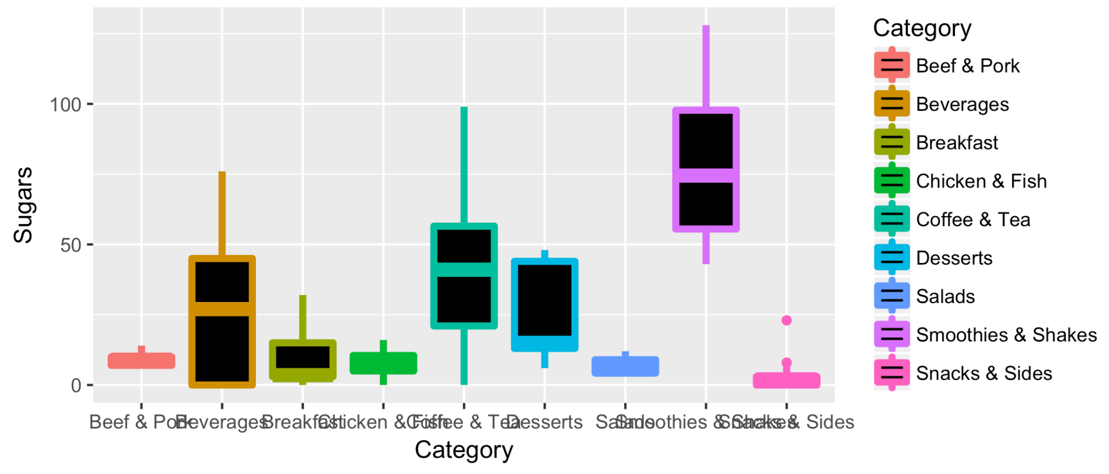
```r
ggplot(data=menu) +
    geom_boxplot(mapping = aes(x=Category, y=Sugars))
```



This plot shows that smoothies and shakes generally have the most amount of sugar. It also shows that beef and pork items roughly have the same amount of sugar while coffee and tea have a wide distribution.

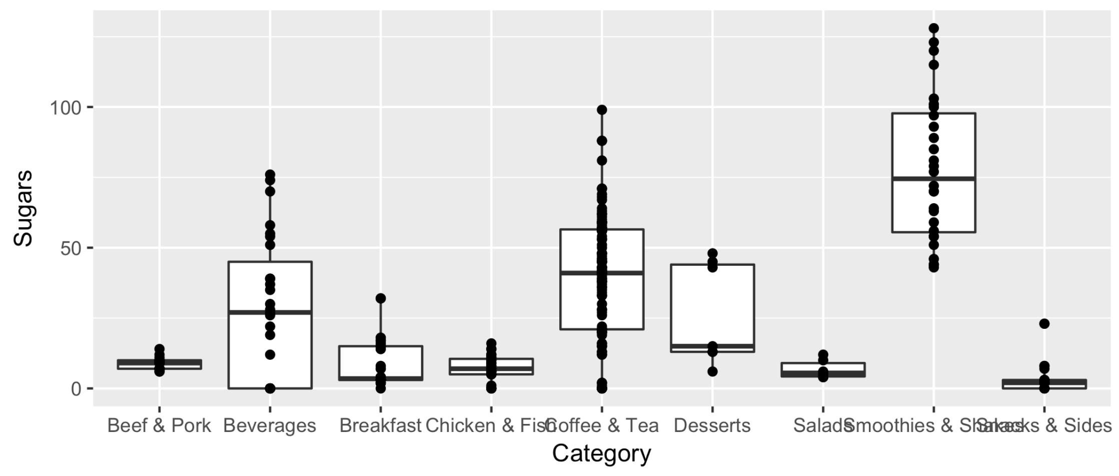
Just like the plots we've seen before, we can modify some of the properties of this plot using the same aesthetics as before (not that you'd want your plot to look like this, but it serves as a good illustration).

```
ggplot(data=menu) +
    geom_boxplot(mapping = aes(x=Category, y=Sugars, color=Category), width=0
.75, size=1.5, fill="black")
```



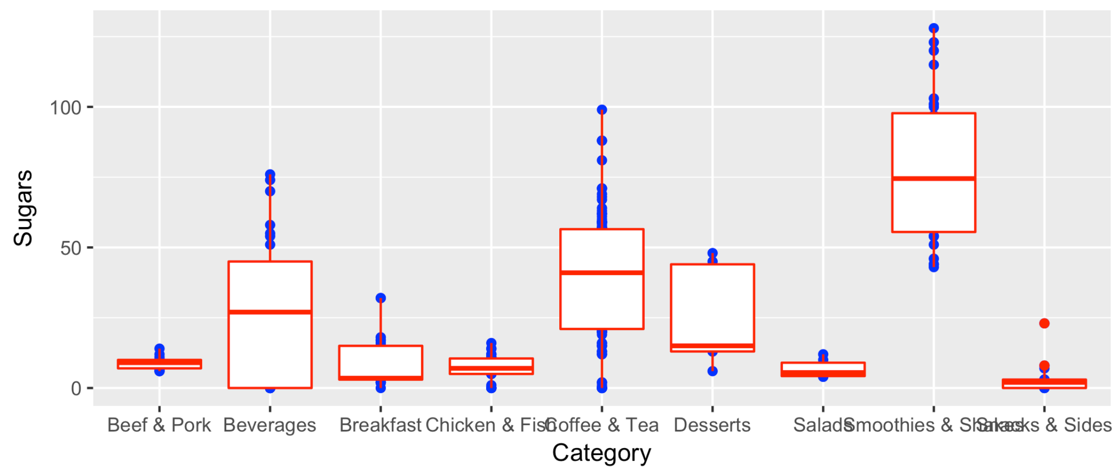
One problem with boxplots is they abstract away from the actual data. It's actually possible to get identical boxplots with wildly different distributions of data. For this reason, sometimes it's nice to plot the points themselves in addition to the boxplot. How can we plot points if we already have `geom_boxplot`. The answer is simple: just do both!

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



The code here is relatively straightforward. We've seen before how we can add a layer to the base gray rectangle by adding a plus sign and then some `geom`. Well we can add as many layers as we want in the same way: just add a plus and then some other ggplot2 function. Here, we're adding a scatterplot on top of the boxplot. The order here is important: the layers are added in the order that they appear in your code: first the base gray, then the boxplots, then the points. So if we switch the latter two around, you'll see that the boxplot covers the points (color added for clarity):

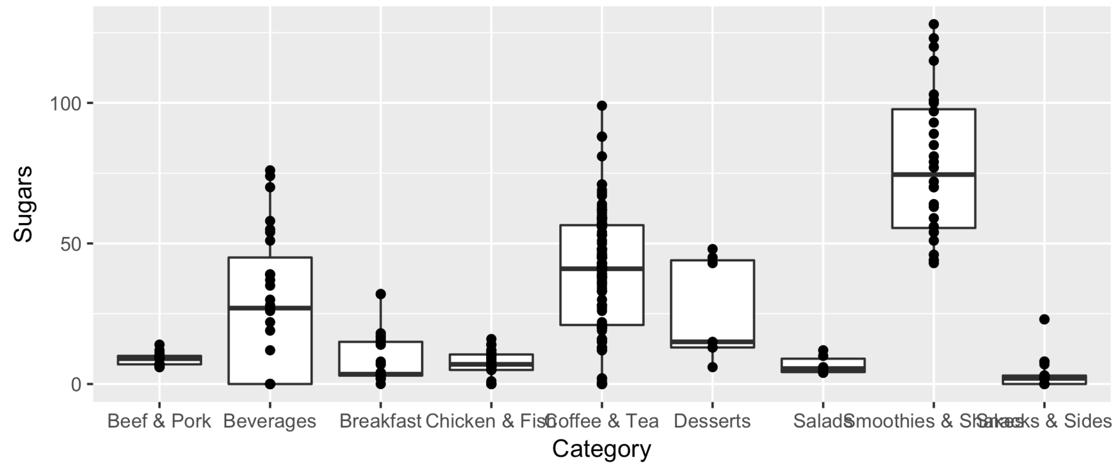
```
ggplot(data=menu) +
    geom_point(mapping = aes(x=Category, y=Sugars), color="blue") +
    geom_boxplot(mapping = aes(x=Category, y=Sugars), color="red")
```



Something else you may have noticed is that there's a bit of repetitive code there. In both `geom`s we need to specify `mapping = aes(x=Category, y=Sugars)`. This is is functional code, but it can be cumbersome. For example, if we wanted to switch from `Sugars` to `Fat`, you'd have to make the change twice. It would be easier if we could somehow combine them both.

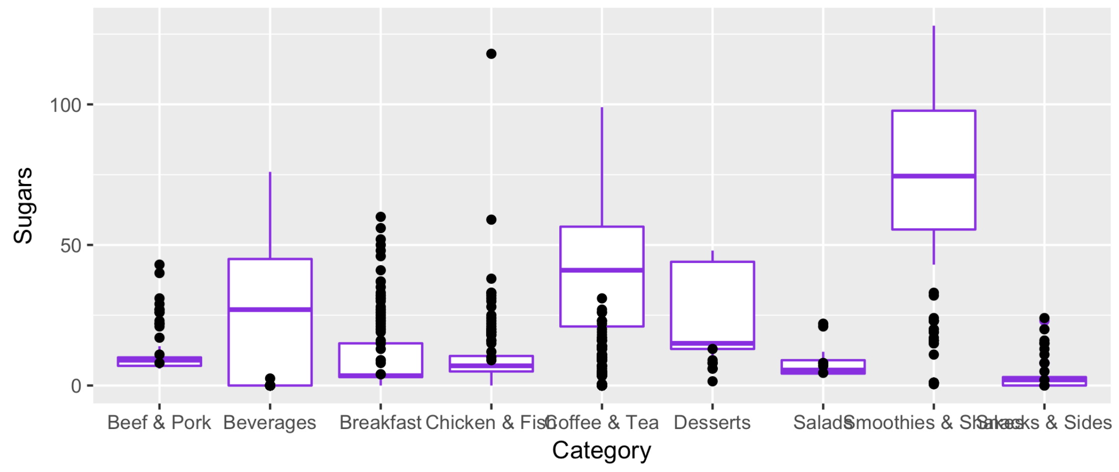
In fact, we can! We can actually move that whole `mapping` part up to the `ggplot` function. Since this is the base layer, it passes that information on to all the subsequent layers as if they were explicitly typed there. In fact, we've already been doing that with `data=menu`. The `data` argument could have been specified in each layer, but we just made it available to all layers by keeping it in the ggplot function. Thus, an equivalent plot as the one with points overlaid on the boxplot, but with cleaner code, might look like this.

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



This saves a bit of typing, but it also makes your code a little bit easier to read as well. We can still have aesthetics and other properties in the geom functions if we want to override what's in ggplot, or specify something that should be in that layer only. In the following chart I specify that only the boxplot should be blueviolet and that the dots should reflect the Fat data instead of the Sugar. (In fact, you can even specify a whole different dataset in individual geoms, but we won't get to that here.)

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

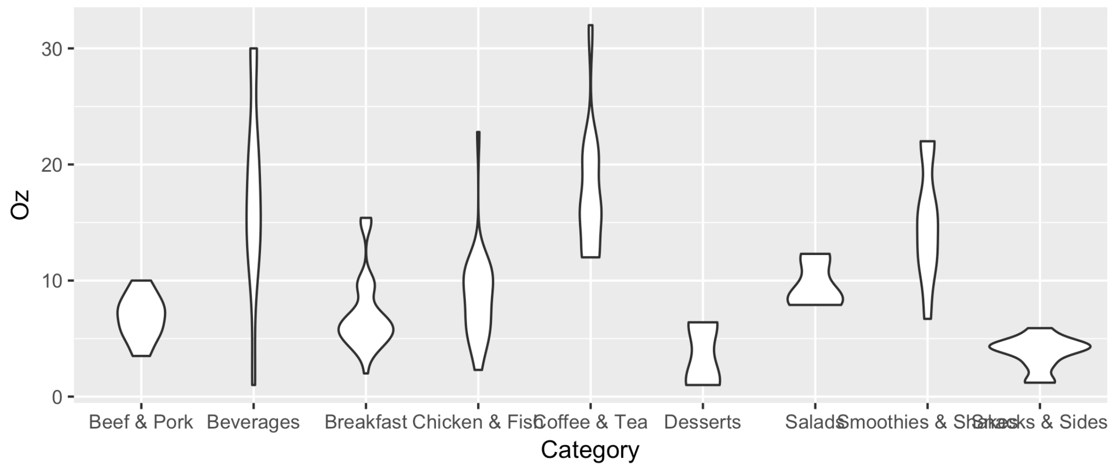


This is not very useful and is an incredibly misleading plot, but it serves as a good illustration of what's possible. Teh layer-by-layer approach to graph building gives you incredible flexilbilty. You'd have a really hard time doing the same thing in Excel.

5.2 GEOM_VIOLIN

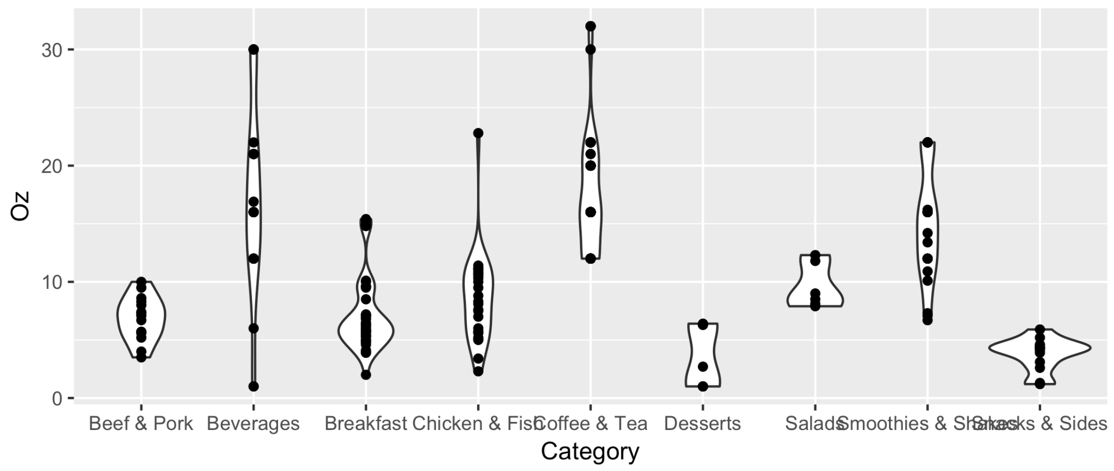
boxplots have been around for a while, and their various components can easily be calculated and drawn by hand. The evolution of boxplots are violin plots, which give a better view of the distribution of the data.

```r
ggplot(data=menu, mapping = aes(x=Category, y=Oz)) +
    geom_violin()
```



The shape of these plots sort of show the distribution of the data. Wider portions mean more data around there. The plot gets its name because sometimes the plots look like violins. You can compare violin plots to the raw distribution by overlaying the points again.

```r
ggplot(data=menu, mapping = aes(x=Category, y=Oz)) +
    geom_violin() +
    geom_point()
```



One thing that is useful about functions in many programming languages (including R and thus ggplot2) is that the name of the argument can be left off. So far in all of our code, we've had to type `data=` and `mapping=` every time. The arguments of a function in R have a default order, so if you know that order, you can leave off the name of the argument. In ggplot2 (and all tidyverse packages actually), the first argument is always `data=`. So if you just type the name of your data frame, ggplot2 will assume that it's the data frame. The second argument in `ggplot2` is `mapping`, so you can leave that off as long as the `aes()` function comes second.

For example, the following two are identical:

```
ggplot(data=menu, mapping = aes(x=Category, y=Oz)) +
    geom_violin(color="blueviolet") +
    geom_point(mapping = aes(y=Fat))
ggplot(menu, aes(Category, Oz)) +
    geom_violin(color="blueviolet") +
    geom_point(aes(y=Fat))
```

Note that for `geom_point`, I was able to leave off `mapping=` because in `geom_point`, `mapping` is the first argument. Even within the `aes()` function, the first and second arguments are `x` and `y`, meaning we can leave those off too. However, in `geom_violin`, I had to specifically say that `"blueviolet"` is the `color` argument, because `color` is *not* the first argument of `geom_violin`. When in doubt, always specify the argument, but it does save some typing when you can leave them off.

5.3 Your Turn!

1. Take a few minutes and play around with boxplot and violin plots for `Sugars`, `Fat`, and `Oz`. Choose one and make a good plot that is faithful to the data. Play around with the aesthetics to make it look good. Be sure to make your code concise yet readable. Keep in mind your audience: if you're going to be the only one seeing this code, do what makes sense for *you*, but if you're going to share the code (and you never know if you will), do what is most readable generally.

2. Think of your own data and what things you could show using boxplots or violin plots. Keep in mind the data types required to make these plots. What kinds of things can you *not* show using these plots?

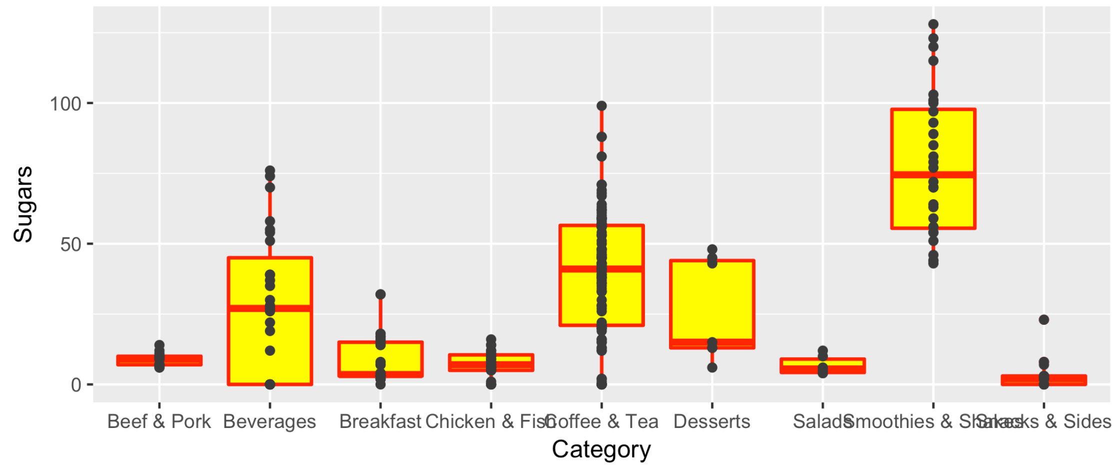
6 Other ggplot2 functions

I can't possibly show you all the ways to visualize your data. Not only is that a lot of ggplot2 but it gets into a lot of statistical background regarding data types and the pros and cons of each visualization type. Data visualization is a substantial subfield of statistics, but it's important to at least know the basics.

Instead, I want to close this workshop by covering just a few more ggplot functions. These are little things that can really help you make your plot your own.

First, what I'll do is actually save a plot as an R object. That way I can just add one line of code at a time to make it easier to see the differences are between these plots.

```r
p <- ggplot(menu, aes(Category, Sugars)) +
    geom_boxplot(color="red", fill="yellow", size=0.75) +
    geom_point(color="gray25")
p
```



6.1 Axes and Titles

You can change the labels of the x- and y-axes of your plot using the `xlab` ("x label") and `ylab` functions. This is useful for lots of reasons:

1. When the columns of your data frame are something abbreviated, you can put the full name in a professional-looking plot. So, you can change `"mpg"` to `"miles per gallon"`).

2. Sometimes your column names are super long. An actual example I've seen is `"education_level_by_number_of_years"`, which could easily be shorted to `"eduacation (years)"`.

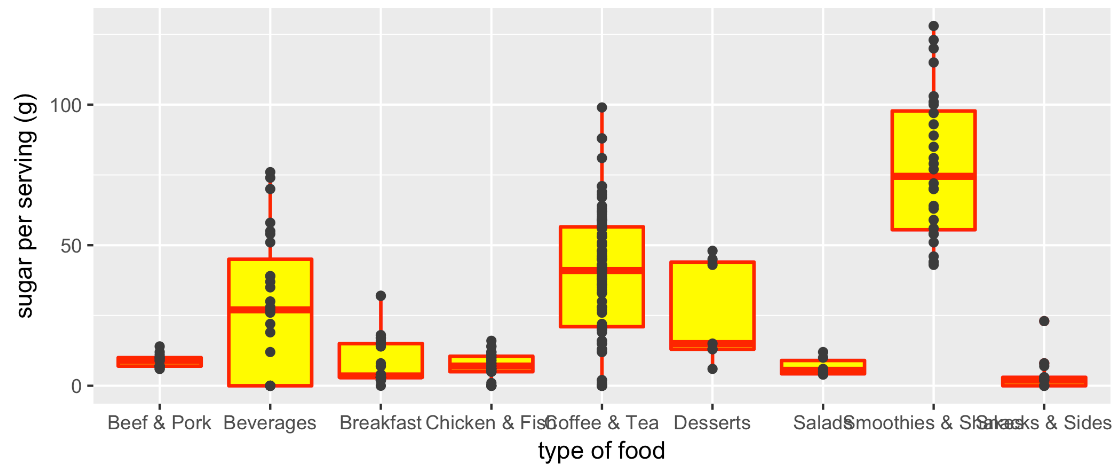
3. Often, all you'll need to do is change from upper-case to lower case or vice versa (`"Sugars"` to `"sugars"`).

4. You may want to add a unit of measurement, so that `"height"` can be changed to `"height (in feet)"`

5. R doesn't like spaces in column names, but you'll probably want them in your plots, as in `"serving_size"` vs. `"serving size"`.

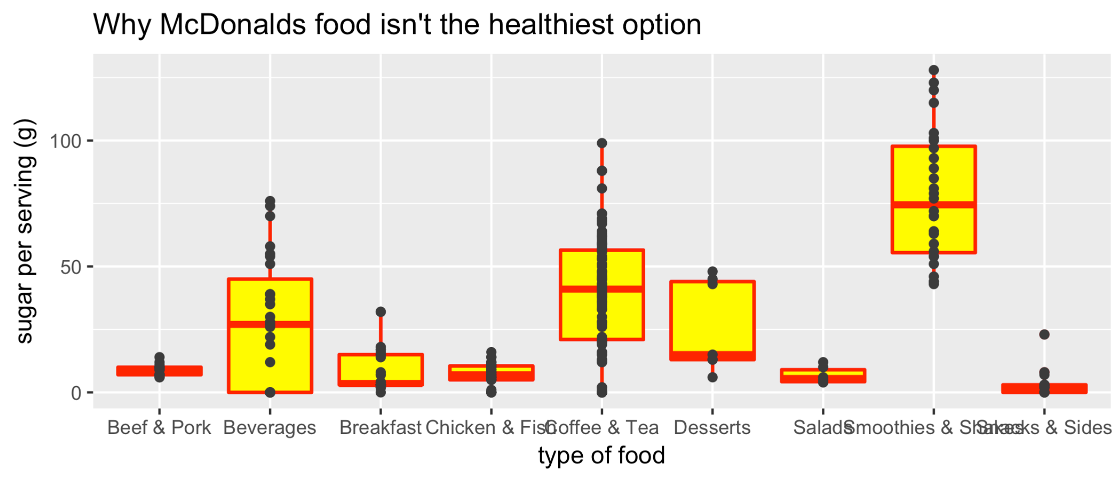
With ggplot2, you can make these changes so that they're reflected in your plot only without having to rename parts of your data frame.

```
p + xlab("type of food") +
    ylab("sugar per serving (g)")
```



You can add a title using the `ggtitle` function. I often place this after my `geoms`.

```
p + xlab("type of food") +
    ylab("sugar per serving (g)") +
    ggtitle("Why McDonalds food isn't the healthiest option")
```



You can actually wrap all three of these into one function `labs`, which gives you the option of adding a subtitle and caption as well.

```
p_with_text <- p +
    labs(x="type of food",
        y="sugar per serving (g)",
        title="Why McDonalds food isn't the healthiest option",
        subtitle="Especially if you want to avoid sugar",
        caption = "Figure 1: Distribution of sugar by category.
        Nice color scheme by the way ;)")
p_with_text
```

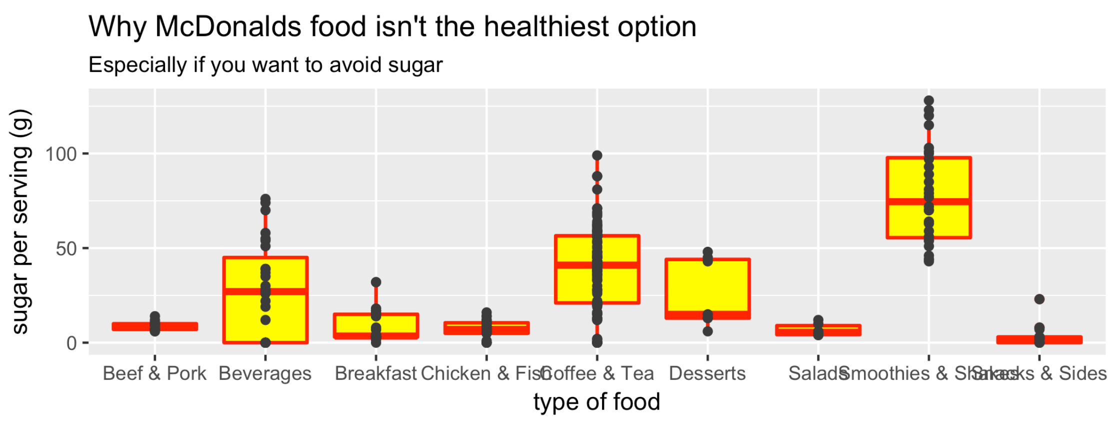


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

6.2 Themes

Something you might be sick of now is the gray background. This can be easily fixed by applying different themes to your plot. I personally am not a fan of the gray, but it does make some parts of the plots stand out more.

The first is my go-to theme. It's simple, and frankly, just isn't grey. This is the black and white theme.

```
p_with_text + theme_bw()
```

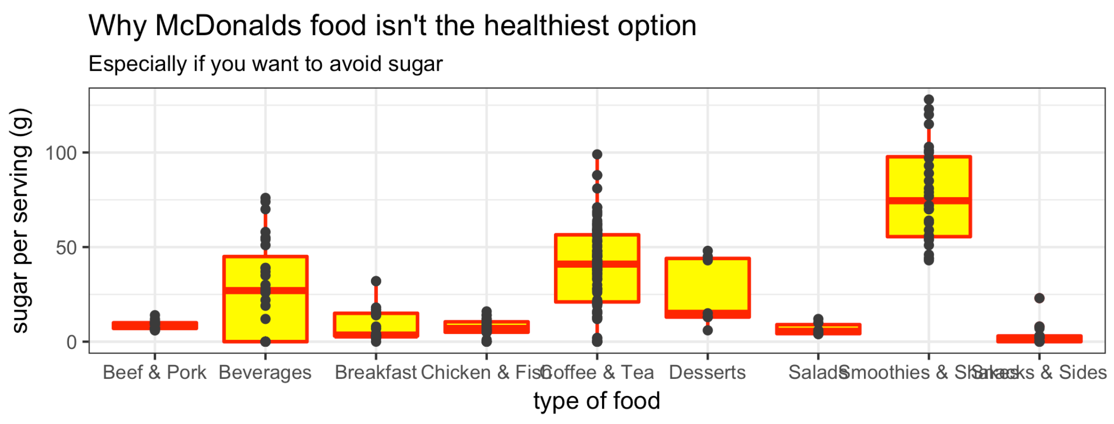


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

The `classic` theme has no grid and the top and right parts of the box are gone too, just leaving the x and y axes.

```
p_with_text + theme_classic()
```

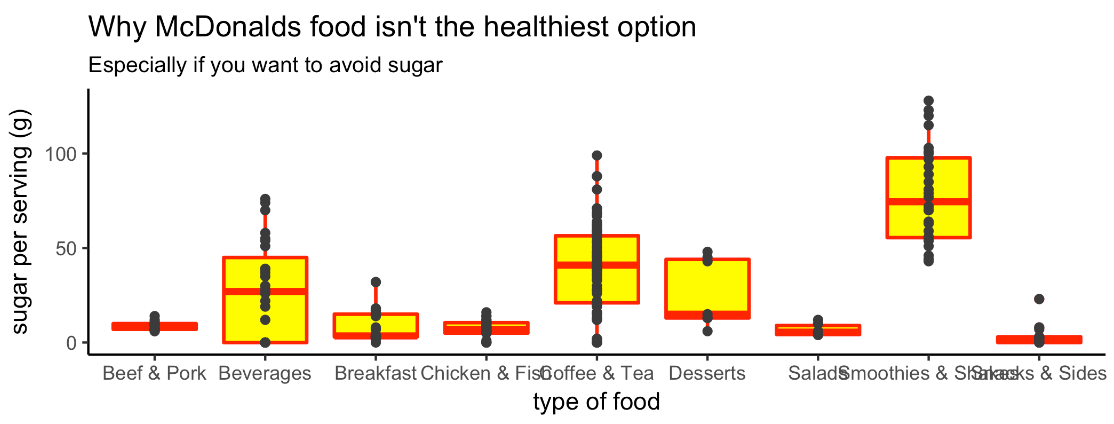


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

The `light` theme is very similar to `bw`. The biggest difference is the outside box is lighter.

```r
p_with_text + theme_light()
```

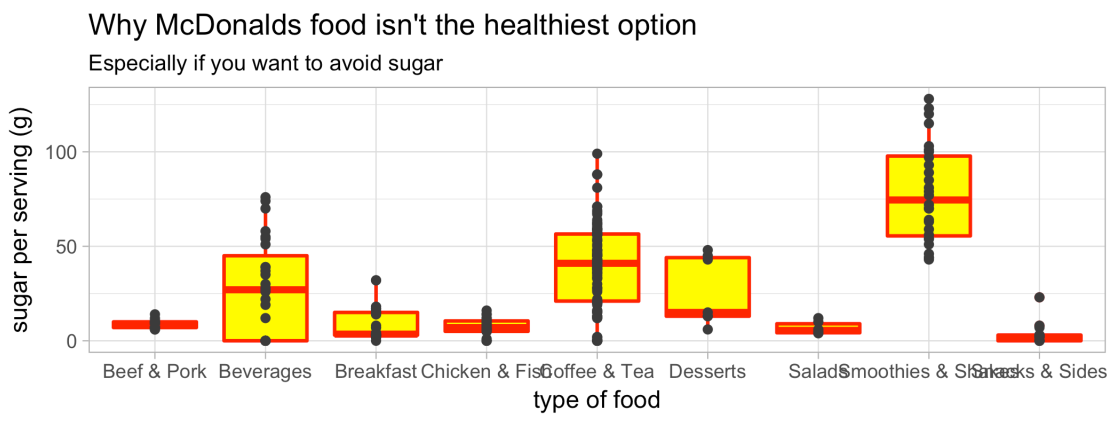


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

If you really like the gray, you can go for the `dark` theme.

```r
p_with_text + theme_dark()
```

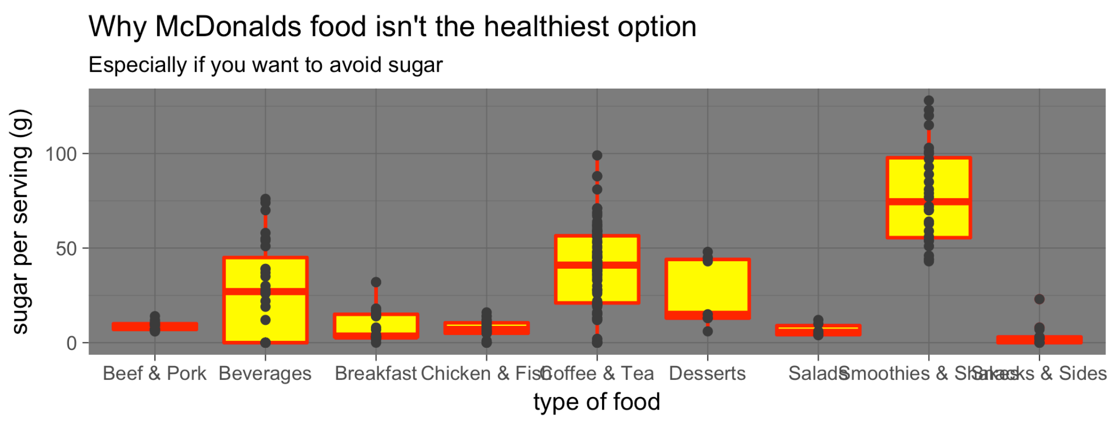


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

In fact, if you're in love with gray, you can set it specifically.

```
p_with_text + theme_gray()
```

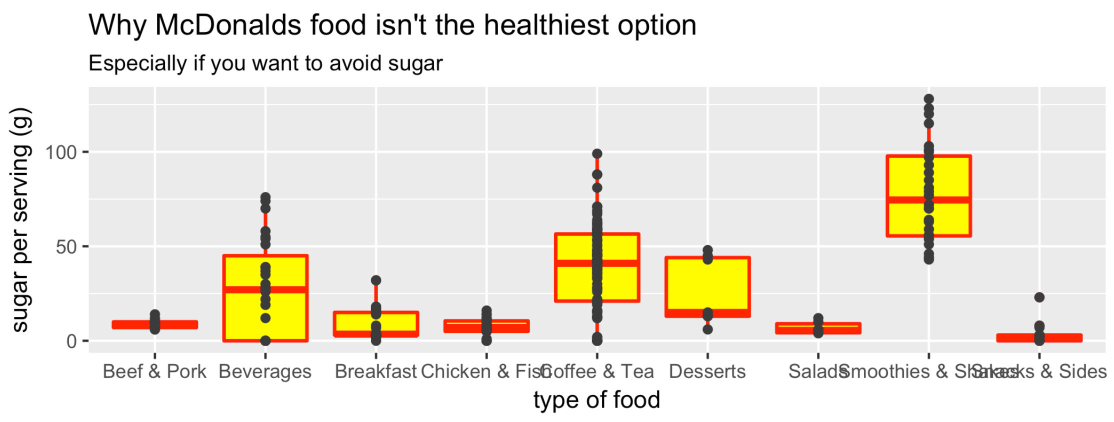


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

The `minimal` theme is even simpler than `light`, but in my opinion it looks a little weird without the outside box while retaining the inside grid.

```
p_with_text + theme_minimal()
```

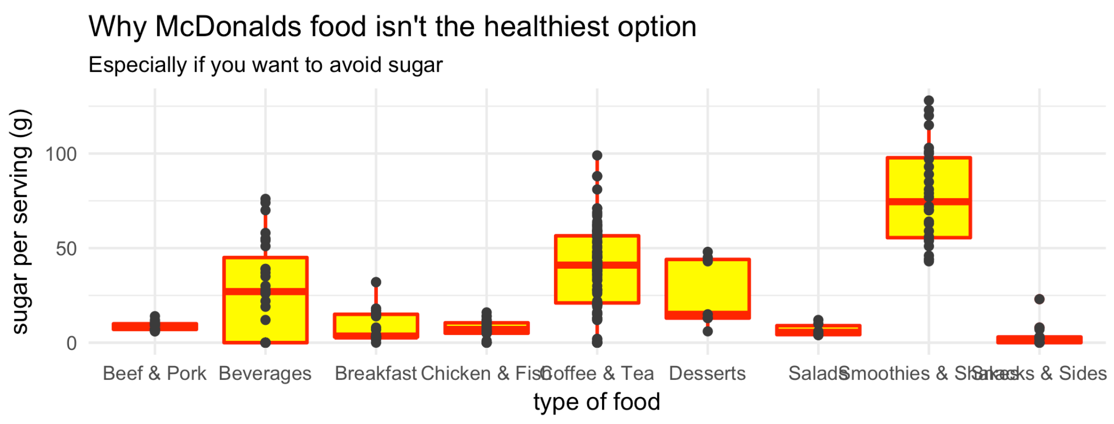


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

Finally, you can go completely `void`, which only includes elements that you specify:

```r
p_with_text + theme_void()
```

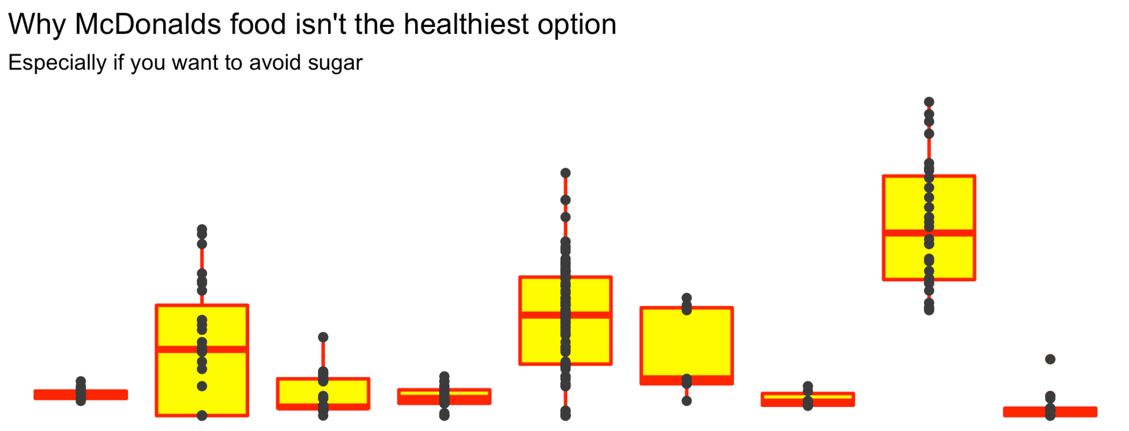


Figure 1: Distribution of sugar by category.
Nice color scheme by the way ;)

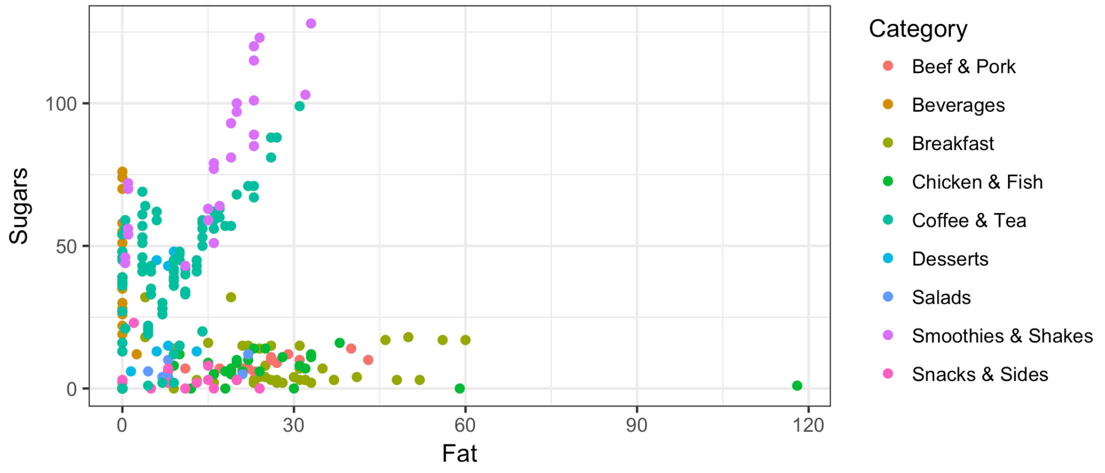
This may seem a little weird, but it does have useful purposes, like when creating maps. Notice that here the *x*- and *y*-axis labels are missing. This is because the `theme` functions override anything else in your plot.

By the way, it is possible to create your own custom themes. I have one that I've made that matches the color and font family of my powerpoint slides so that the plot seamlessly integrates with the slides. I can't cover that here, but I have written a brief blog post about it, which you can read here.

6.3 Faceting

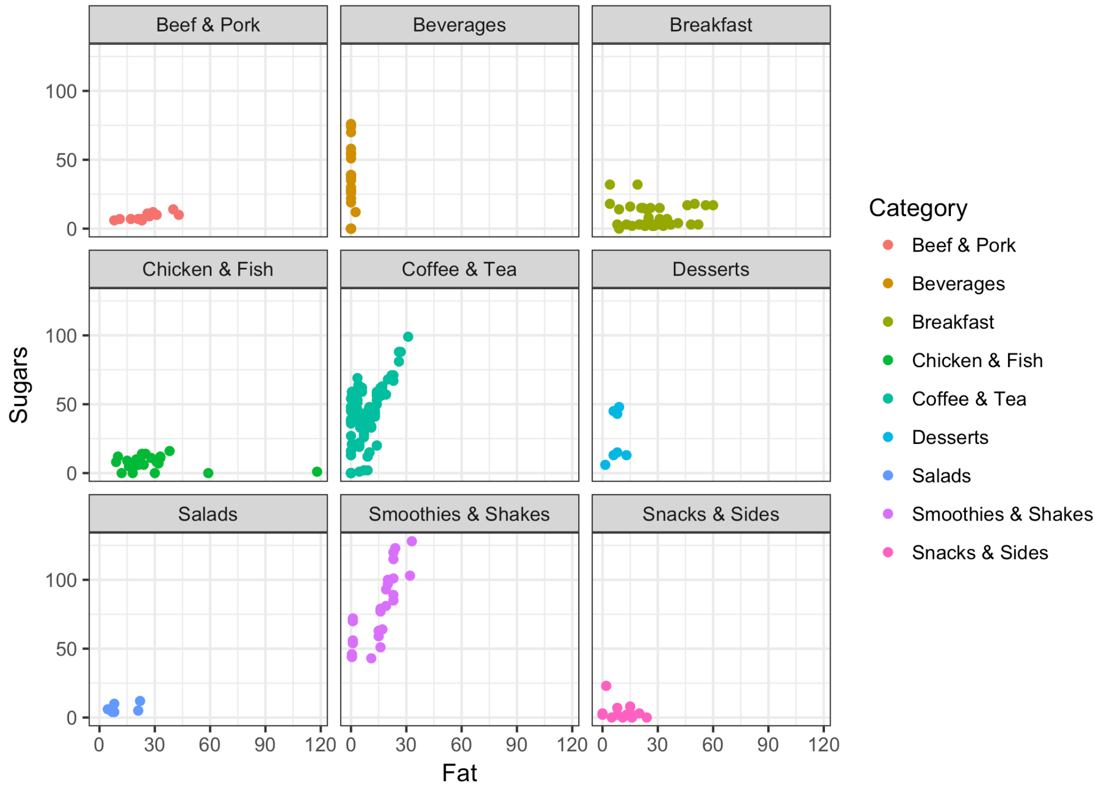
For our last topic, I thought it might be useful to look at faceting. As we've seen, ggplot is smart and can do a lot of stuff with very little code. One of these things is to create separate plots for each category. Let's say we want to plot fat and sugar. We've learned how to do that.

```r
ggplot(menu, aes(Fat, Sugars, color=Category)) +
    geom_point() +
    theme_bw()
```



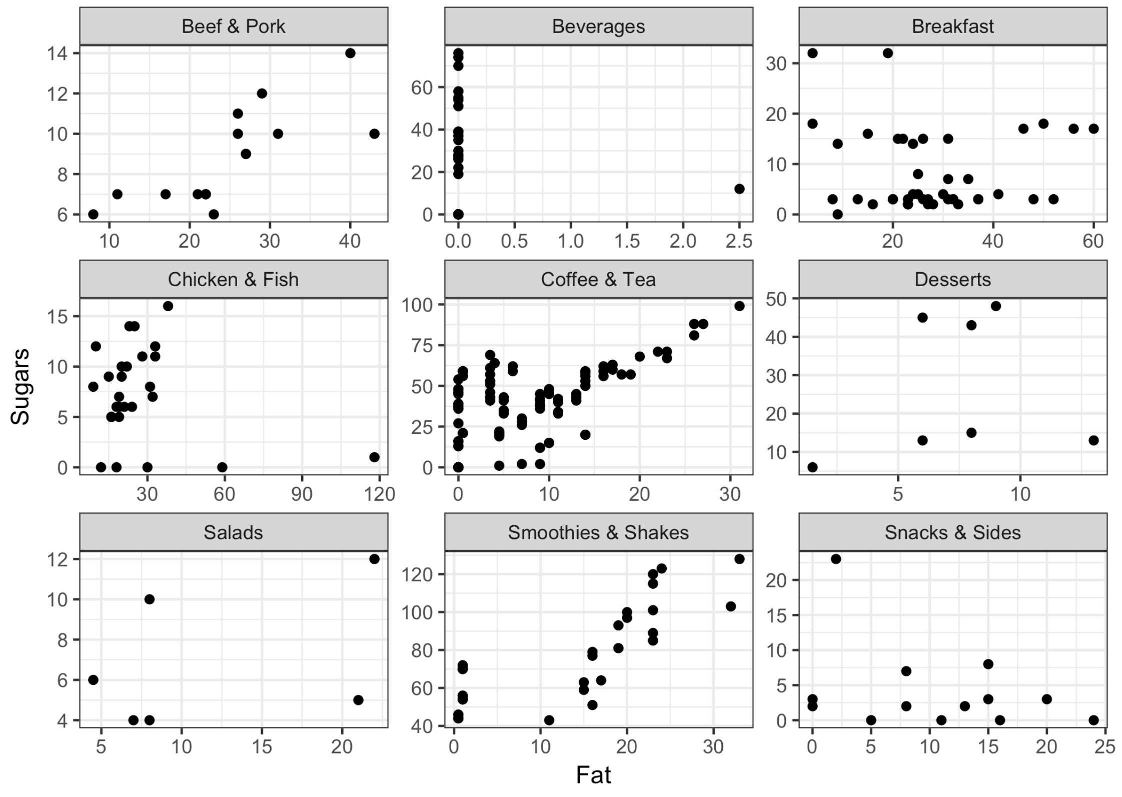
Sometimes though, it might be helpful to split that up into one plot per category. We can do that with `facet_wrap()`. The argument that this takes is the name of the column you want to split it up by, preceded by a tilde (~). This tilde is there because it actually means it's a function, but we won't worry about that.

```
ggplot(menu, aes(Fat, Sugars, color=Category)) +
    geom_point() +
    facet_wrap(~Category) +
    theme_bw()
```



(This would be really hard to do in Excel, by the way, because you'd have to create 9 plots instead of just one.) By default, the axes are identical for each plot, which makes it useful to compare each group. Sometimes it's helpful to let each plot have its own axes so you can examine the trends within categories. We can do that with the addition of `scales="free"` as an argument to `facet_wrap()`. We'll also turn the colors back to black since each color has its own plot now.

```
ggplot(menu, aes(Fat, Sugars)) +
    geom_point() +
    facet_wrap(~Category, scales="free") +
    theme_bw()
```



This is a very different way of viewing the data. Facet wrapping is a powerful way of splitting your data up, so it's a good one to have in your toolbox.

7 Conclusions

This workshop can only cover so much, but I've tried to show lots of different things that can be done with ggplot2 without a lot of work. Before we finish, it's nice to see where you can go to learn more.

7.1 Where to go for help

I'll be the first to admit that working with ggplot2 is rough and there is a learning curve. However, after a while, especially once you've figured out what kinds of plots you tend to do with your own data, things come naturally. In the mean time, here are some places that I have found to be handy bookmarks when struggling with ggplot2, many of which are produced by Hadley Wickham himself.

1. Springer has published a book in their "Use R!" series that's simply called *ggplot2*. It's written by Wickham and is basically the full documentation for the package. It's comprehensive but still aimed at learners. UGA students can download the eBook for free through the library.

2. There's a cheatsheet available by going to RStudio -> Help -> Cheatsheets -> Data Visualization within ggplot2. It's not exactly easy to learn from it, but it's a good reminder of things you've learned already.

3. There's always help within R itself. You can type `?geom_point` for example and get help on that function.

4. ggplot2's official website, ggplot2.tidyverse.org, is a good launching pad. From there you can find additional resources, tutorials, and presentations that Wickham has done. Two such resources can be found here and here.

5. One great site that appears a lot when I google for help is cookbook-r.com. In particular, they have a section on graphs, which mostly covers ggplot2. There, you can see how to manipulate things like colors, text size, legends, axes, and all sort of other stuff. It's probably the resource I used the most when learning ggplot2. They have lots of examples of the many things that are possible to manipulate, showing that you really do have control over every aspect of the plot.

7.2 Final remarks

Yes, ggplot2 takes a bit of time to get used to, but it is worth it. It is the best way that I know of to create clean, professional visualizations. It's free, and there's a ton of help online. It's also completely customizable.

I hope this workshop has at least got you interested in using ggplot2. I know there's a lot to learn, but there are patterns in the code, making it somewhat easy to try new things. Visualizing data is a key component of data analysis: sometimes it's for your own consumption so that you can understand your own data and sometimes its for public consumption such as in a presentation or a paper. ggplot2 is a very good solution that should solve most people's needs and with it you can soon be able to create stunning visualizations with your data.