Week 2: Data Visualization with ggplot

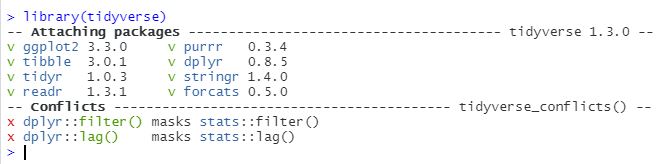
This week, we’re diving right into making plots with ggplot**,** a member of the tidyverse that we installed last time. Now, if you do a bit of googling about data visualization, you’ll quickly find that people have a LOT to say about it. There are lots of different philosophies, best practices, arguments… it’s fascinating stuff, but I’m not going to even try to go into all that. Coming from a scientific background, I have my own opinions about what makes a “good” visual, and I’m sure you all have your own thoughts. The authors of *R4DS* have included a link to an article about the theory of ggplot (<http://vita.had.co.nz/papers/layered-grammar.pdf>). If you’re interested, go ahead and give it a look! But for now, I’m going to focus on how to actually use ggplot.

A quick note before we get started: there’s going to be some stuff we do that isn’t going to make a ton of sense, at least not at first. That’s the downside of starting with data vis versus building the foundations first… but with some googling we can figure most everything out. Now, onward!

To do anything, we’ll need to load up the tidyverse like before, since that’s where ggplot lives:

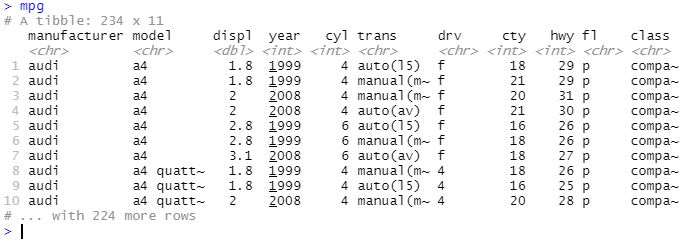
> library(tidyverse)

If you have it installed correctly, you’ll see this:



(The “conflicts” part gave me pause, but this is normal apparently).

Now we can start asking some questions! While we will presumably be learning to load and clean outside data in future chapters, for now we will take advantage of that fact that ggplot comes pre-loaded with some clean practice dataframes. I think about dataframes as kind of like spreadsheets or tables. They have two dimensions, with columns of variables and rows of observations. The first dataframe we’re working with is the mpg dataset. To see what it looks like, just type mpg into the console, and this is what you should see:



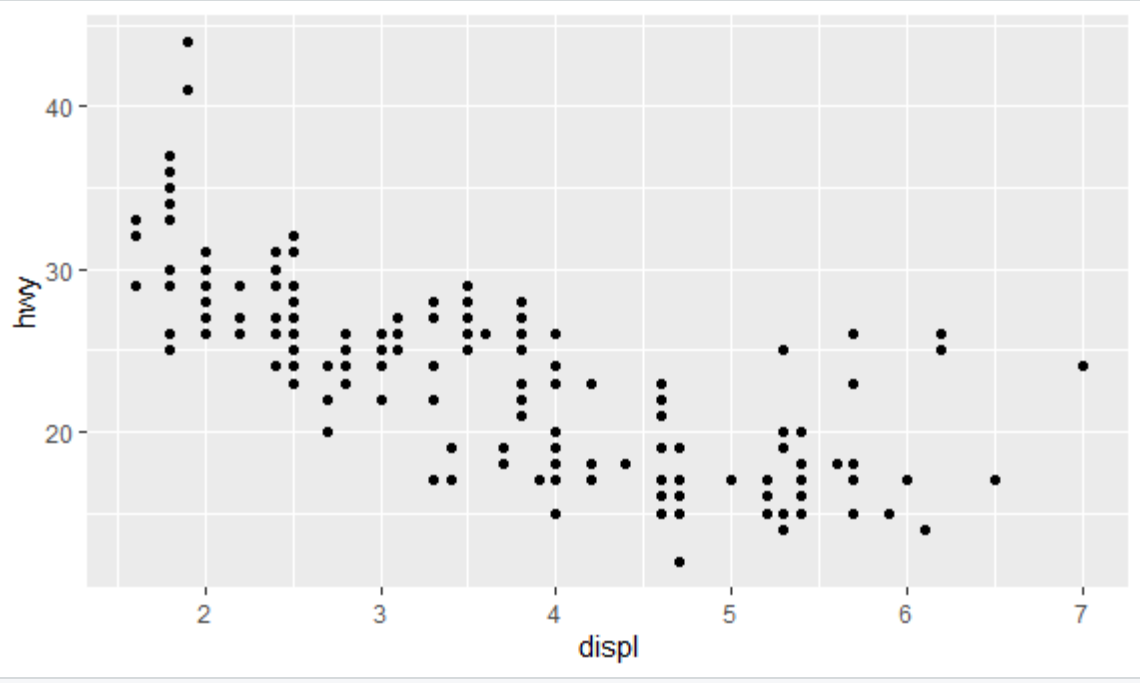
There are 11 variables, or fields, and 234 observations in this dataset. If you type ?mpg into the console, you can get some more information on each field.

The first plot *R4DS* has us make begins with typing this line into our console:

> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y =

hwy))

Which shows us this scatterplot:



Going through the syntax: ggplot is the function (note that we don’t include the 2 here, the package is called ggplot2 but the function we call is ggplot). The part in parentheses (the “argument”) specifies what data you are using- the mpg dataframe in this case. This is the first layer. If you just run ggplot(data=mpg), stuff will happen, but all you’ll see is a gray box. Not super useful. The plus sign means we’re adding a layer.The geom\_point() function specifies that we’re making a scatterplot. It’s argument is *mapping* (sensing a pattern?), which basically just says “this is where I’m specifying how my data will look.” The next bit, *aes* (or “aesthetic specifications”) is where you’ll actually *specify* the *aesthetics* (in this example, all we specified is that *displ* (engine displacement, which is a proxy for engine size) is on the x axis and that *hwy* (highway miles per gallon) is on the y axis). In fact, as we go forward, we can use this basic template for any geom, or “geometric object,” changing the dataset, geom function, and/or mappings as we go. So, our general template will look like this:

ggplot(data = <DATA>) +

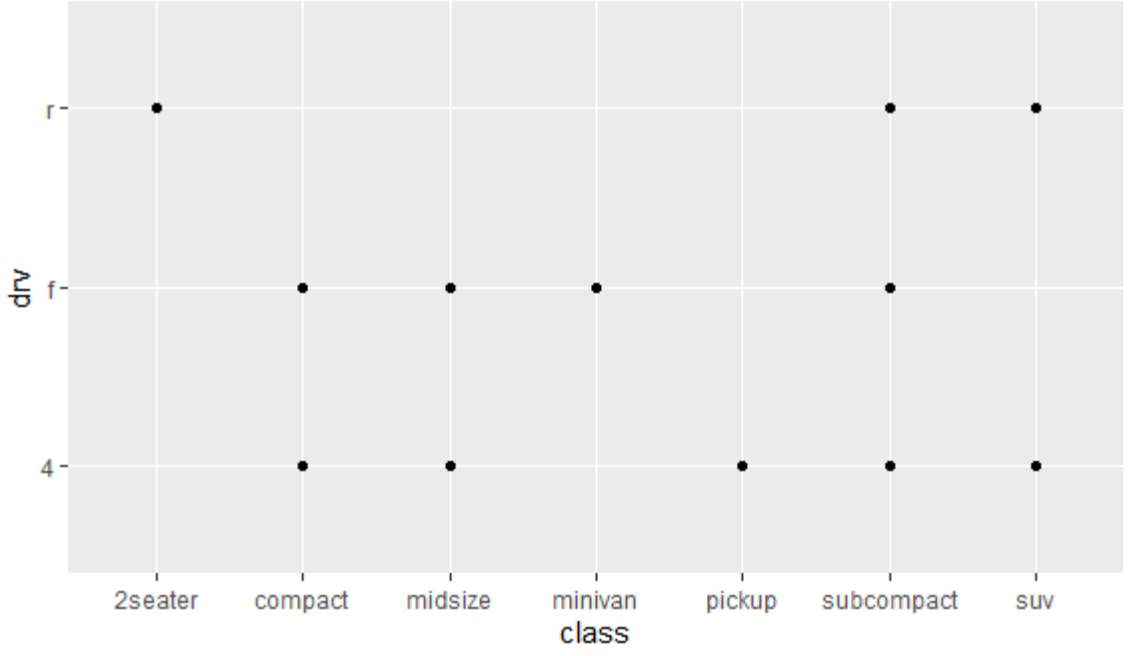
<GEOM\_FUNCTION>(mapping = aes(<MAPPINGS>))

You can choose any two variables for your scatterplot, but some might be more useful than others. Let’s look at exercise #5 for this section. The authors ask us to look at the scatterplot of *class* vs *drv* (type of drive train).

> ggplot(data = mpg) +

geom\_point(mapping = aes(x = class, y = drv))

You should get something like this:



There’s not much we can get from this, because *class* and *drv* are categorical rather than continuous. There are ways to plot categorical data in informative ways, but we’ll see more about that later.

What’s fun about mappings in ggplot is that you can map lots of other aesthetics besides just telling you what’s on the x or y axes. Some of them just make your chart look nice. Others, however, can reveal important information about your data!