Grammars of visualization

Our readings this week looked at a wide variety of forms of data visualization.

This week we're going to look at one particular engine: `ggplot`, one of the more influential graphical engines

The grammar of graphics: Declarative Programming

Bear with me, this is going to get a little technical.

There are a lot of different types of computer languages out there. One of the most basic distinctions is between *declarative* and *imperative* programming. Some languages are more declarative and others more imperative.

Imperative programming means giving the computer step-by-step instructions on what to do. Move the cursor to this point on the screen; cycle through a list; load a shapefile into memory, filter it down to the USA, and draw a map. To do imperative programming well, you have to understand how the code works on the computer as a physical machine with files, processes, and memory.

Declarative programming takes place a higher level. In declarative programming, you tell the computer not *how to do something* but *what you want done*. The software then figures out how to do it for you.[1]

At the end of the semester we'll be looking at HTML, which is perhaps the most successful declarative language in existence. It doesn't write programs: it describes what text should look like, and the computer figures out how to make things bold, or linked, or where to insert an image.

If there is a declarative language out there suited for your task, it can both be easier to use and much more powerful than mucking about in computer programming as a process. Fortunately, there is an excellent declarative language for making attractive charts: the `ggplot` package for R by Hadley Wickham. The "GG" in "gggplot" stands for "Grammar of Graphics:" this is a formal way of thinking about graphics that is both influential in the world of data visualization and almost as easy to use as something like Microsoft Excel, but considerably more powerful.

Let's make a plot

`ggplot` is part of the larger R environment for statistical computing. This is a full-fledged programming language, but we won't be touching on most elements of it in this class.

Before the workset session, those with computers should install RStudio (the program you should use to run R) and install ggplot2. That should involve typing the following commands into the line on the left side of the screen when you install R:

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

Just to make sure things work, trying pasting in some lines of code:

```
2+2
tolower("HeLLO EverYOne")
```

1. This might sound like like imperative programming is real programming, and declarative is "softer." That's not really the case, because all languages are fundamentally abstractions on the physical machine. A few people as an abstract exercise will spend time writing machine code in a format that directly works with the architecture of the underlying chips, but this is almost never worth the time.

Loading in data To make a plot, we have to just spend one brief minute in declarative land to load in the data. Cut and paste this code to load it.[2]

```
ships = read.csv("http://benschmidt.org/whalers.csv")
```

Now we can use ggplot to make charts.

The simplest of all charts are *histograms*. This is a fancy name for a particular type of bar chart: one where the y axis is just a count of observations. If you ask for a barplot in ggplot, you'll get it.

In ggplot, we build this up by adding plot elements. One of the most basic questions might be: what years do we have here?

We do this with three parts:

1. By entering the ggplot sublanguage on our plot with `ggplot(ships)`
2. By giving an aesthetic mapping: `aes(x=year)` says that the x axis should represent the year.
3. By giving the plot type: we add `geom_bar()` to the end.

```
ggplot(ships) + aes(x=year) + geom_bar()
```

We can make these histograms for any variable: for instance, "rig" is the type of ship. Don't cut and paste this: instead, just press the up arrow to go back to the previous element in your code.

```
ggplot(ships) + aes(x=Rig) + geom_bar()
```

There are lots of ways to "escape from flatland" here. One is to expand the aesthetics so we show *both of these at once*, by using the "fill" aesthetic.

```
ggplot(ships) + aes(x=year,fill=Rig) + geom_bar()
```

Remember, these counts are of *people*, not ships. So we can ask questions about the individuals, as well.

The second most basic plot is a scatterplot. We represent those as points.

```
ggplot(ships) + aes(y=height,x=Age) + geom_point()
```

Boxplots are another basic form.

```
ggplot(ships) + aes(y=height,x=Rank) + geom_boxplot()
```

2. Seriously, cut and paste. Don't waste time typing it in: you'll make a typo and not see why it fails, and get frustrated. One of the principal mistakes humanists make in computer programming is trying to maintain standards of intellectual propriety that come from writing. Far more important than understanding exactly what's going on is being willing to trust other code and change it on the margins.

Finally, and perhaps most specially for Tufte fans, you can create faceted charts: what Tufte calls **small multiples**. The way to do this is `facet_wrap`.

```
ggplot(ships) + aes(x=year) + geom_bar() + facet_wrap(~Rig)
```

`facet_grid` lets you do 2-dimensional grids. Here is decade and month, by amount. It's too many!

```
ggplot(ships) + aes(x=year) + geom_bar() + facet_grid(month~decade)
```