Visualizing Data: the basics

Ben Schmidt

2019-01-31

Why Visualize?

Graphic presentation is an increasingly important element of humanities scholarship. For exploratory data analysis, visualization has long been seen as the most useful way of working with data. As we saw with the information in "Anscombe's quartet," it is quite frequent that patterns which can be quite difficult to describe mathematically are immediately obvious when plotted with the proper specifications.

Visualization is also an important element of scholarly **communication**. As scholars like John Theibault, etc. have recently argued, visualizations can themselves be an element of historical arguments. (As we'll read next week, there are reasons to be concerned that visualizations may interpose scientisitc forms of reading).

This work is the heir to a long history of non-lexical representations of history. Graphic visualization is occasionally described as an offshoot of the sciences. In fact, the tradition is probably more closely connected to the historical and social sciences than to any of the "hard" sciences: the early history of data visualization and thematic cartography are mostly attempts to bring visual order to social and historical systems.[1]

The question of whether an image has "an argument" is a thorny one; the question of whether an image can *provide evidence* should be much less so.

The Grammar of Graphics

In `dplyr`, we have already learned a "grammar of data analysis." It provides a variety of functions for describing transformations you wish to make to data; to combine them, to aggregate, to filter, and so forth. The *content* of those transformations is left up to the user; this is why Wickham uses the metaphor of a "generative grammar" to describe it. A grammar gives rules about construction that make it possible to describe operations much more easily than might be otherwise possible.

> Historical note: this idea of a generative grammar as a live influence in general academic culture goes back to linguistics and particularly Noam Chomsky, who made his first career describing the underlying rules of language. (Chomsky, in addition to his current role as a critic of modern capitalism, continues to defend his theories inside linguistics; his 2011 exchanges with Peter Norvig, the chief research scientist at Google, over the validity of the knowledge derived from machine learning techniques are well worth reading.) The insights were useful outside linguistics: the conductor Leonard Bernstein gave the Charles Eliot Norton lectures on aesthetics at Harvard in the early 1970s by applying Chomskyian techniques to

This idea of a grammar was most influentially described in the statistical literature in 199X by Leland Wilkinson in his book "the grammar of graphics." We'll be exploring data visualization, at first, in the context of an implementation of that grammar also by Hadley Wickham.

The name of this package is `ggplot2`. (`gg` for "Grammar of graphics": `2` because it is version 2 of the software.) I'll be referring to it as simply `ggplot`, because that is the name of the function used to create the core object.

[1] Grafton and Rosenberg; Playfair; Friendly; DuBourg.

The philosophy of ggplot

ggplot uses a philosophy of chaining that should be familiar from your work with dplyr. But while dplyr is a chain in which order is paramount, ggplot is a little more flexible. You have to create a plot using the ggplot() function: but then you can add elements to it in an order that's not always strictly proscribed. But in creating a plot, you'll certainly have to do each of the following (though sometimes just implicitly):

1. Selecting a data source:

The function ggplot generally takes as an argument a single element. For example: if we want to plot the changing urban population of the United States, we can use functions from dplyr to collect it. Then we can create an object base_plot that contains the beginnings of the plot.

```r
library(dplyr)
library(magrittr)
library(ggplot2)
populations =
  read.csv("../data/CESTACityData.csv") %>%
  gather(year, population, starts_with("X")) %>%
  mutate(year = year %>% str_replace("X", "") %>% as.numeric)

totalpopulations = populations %>%
  group_by(year) %>%
  filter(population>0) %>%
  summarize(USpop = sum(population))

base_plot = ggplot(populations)
```

2. Choosing a "geometry" (or plot type)

The next step is choose a particular way to plot the data. This is called a "geometry" in ggplot, and it corresponds roughly to the type of plot that you'll be using. There are many different types of plots that make sense: line charts, bar charts, maps, histograms, and so forth.

For this example, we'll make a simple line chart.

```r
base_plot + geom_line()
```

This still doesn't work, because the program doesn't know *what it's plotting.*

3. Setting an aesthetic mapping.

As Lev Manovich says, plots work by reducing real elements to abstractions. The "aesthetic" component of a ggplot graph is the list of abstraction you want to enforce. Each particular geom will require or use certain different types of data. A line chart plots one thing on the x axis, and another on the y.

We set this in ggplot using the function aes. aes specifies the aesthetic mapping. So make a linechart of population over time, we simply tell it that y should be population, and x should be time.

You may need to type as.numeric(year) instead of year.

```r
ggplot(totalpopulations) +
  geom_line(colour="blue") +
  aes(x=year,y=USpop) +
  geom_point(size=10,alpha=.6,color="blue")
```

4. Setting scales

Scales are some of the most important elements of any chart. For humanities data, in most cases a logarithmic distribution fits the data better than a straightforward linear one. (Log distributions assume that rather each tick of the axis indicating an increase of a fixed amount,

```
ggplot(totalpopulations) +
  geom_line() +
  aes(x=year,y=USpop) +
  scale_y_continuous(trans="log")
```

Other sorts of scales will take other options. For example, a color scale might benefit from manually specifying the colors that you're interested in.

5. Setting Facets.

Edward Tufte extols "small multiples." In ggplot, this is possible through the `facet` functions: `facet_grid` (when you have both x and y aspects to facet off of) and `facet_wrap` (when you only want to facet off a single thing.)

Our chart of US urban population can accomplish the same thing for each individual states by first grouping by state in the creation field, and then by faceting by state. Try adding "color" to the aesthetic on one of the variables to make it more attractive.

```
populations %>%
  group_by(ST,year) %>%
  filter(population > 0) %>%
  summarize(pop = sum(population)) %>%
  ggplot() +
  geom_line() +
  aes(x=year,y=pop) +
  scale_y_log10() +
  facet_wrap(~ST)
```

6. More esoteric settings

1. Choosing coordinate systems.

For the most part, you'll probably be plotting in cartesian space–that is to say, your coordinates will look like graph paper.

Cartesian is not the only coordinate system, though. Two other particularly important ones built into `ggplot` are `coord_polar` and `coord_map`.

- `coord_polar` uses polar coordinates: instead of positioning an element by x and y, it positions it by angle and distance. Some quite famous charts use it, such as Florence Nightingale's Coxcomb charts.
- `coord_map` allows you, with the `mapproj` package, to make a wide variety of maps in R that project latitude and longitude in ways other than the standard projection system.

Because it will form the basis for further work, here is code to plot cities by

```
cities = populations %>% filter(year==1950)

cities %>%
   filter(LON < 0) %>%
  ggplot() +
```

```r
   aes(x=LON,y=LAT,size=population,color=ST) +
   geom_point()
```

2. Labels

Charts should have explanations

The `labs` function lets you specify those.

```r
cities = populations %>% filter(year==1950)

cities %>%
   filter(LON < 0, ST != "AK", ST != "HI") %>%
  ggplot() +
   aes(x=LON,y=LAT,size=population,color=ST) +
   geom_point() + labs(title=("The eastern half of the United States is more populated citywise."))
```

3. Themes.

You can specify the appearance of grid lines, legend position, and all sorts of other elements through the theme argument. Online documentation may be necessary for the time being.

Exploring different geoms

We're going to explore these geoms with a cleaned version of the `crews` dataset.

```r
crews = read_csv("../data/CrewlistCleaned.csv")
```

Histograms

For statisticians, the most basic chart is a histogram, which shows how frequent a single variable is at different levels. We can plot the height of our sailors by adding a new *layer* to the plot: that consists of the function `geom_histogram` to build a histogram, and the aesthetic `aes(x=height)` to tell it we want a histogram about the distribution of height.

```r
ggplot(crews) + geom_histogram(aes(x=height))
```

That might seem like a lot of work, but the advantage is that once a plot is created, we can simply swap out the aesthetic to plot against–say–`date` or `age` as well.

```r
ggplot(crews) + geom_histogram(aes(x=as.Date(date)))
ggplot(crews) + geom_histogram(aes(x=Age))
```

Barplots

When you use a histogram with a *categorical* variable, you ask for a barplot, as when we look at the types of ships in the sample.

```r
ggplot(crews) + geom_bar(aes(x=Rig))
```

A barplot is different, though, because we might want to add some more variables in. For example, we can add another aesthetic for 'fill' (which gives the color of the bars):

```r
ggplot(crews) + geom_bar(aes(x=Rig,fill=Rig))
```

That's a nice chart. But we could also change it so the x-axis contains to information by just setting it to an empty string, and the bars will appear stacked on top of each other.

```r
ggplot(crews) + geom_bar(aes(x="",fill=Rig))
```

That's not as good a way of visualizing the information: you have to compare the size of chunks against each other. But if we make one more tweak—setting the y axis to a polar coordinate system—suddenly it becomes very familiar:

```r
ggplot(crews) + geom_bar(aes(x="",fill=Rig))

ggplot(crews) + geom_bar(aes(x="",fill=Rig)) + coord_polar("y")
```

It's a pie chart! Since Edward Tufte, pie charts are universally reviled; the grammar of graphs is describing them here as "a stacked bar chart plotted in a polar coordinate system."

Scatterplots

For exploratory data analysis, the scatterplot is to multivariate data what the histogram is to single-variable data.

Summary stats are useful, but sometimes you want to compare two types of charts to each other. The next basic chart to use is a scatterplot: in ggplot, you can get this by using "geom_point." Suppose, for example, we want to compare height against age.

```r
ggplot(crews) + geom_point(aes(x=Age,y=height),alpha=.1)
```

```r
ggplot(crews) + geom_point(aes(x=Age,y=height,color=Rig))
```

Density Plots

In this case, that doesn't give you all that much useful: but it points you to another function, "stat_density," which itself points to "density," the basic function: there you can see that 'adjust' is what sets the smoothing bandwith.

```r
?stat_density
```

Just do some survey plots on the data we have. (You might need the hexbin package for this)

```r
ggplot(crews) +
  geom_point(size=1,alpha=.2,position=position_jitter(h=1/12,w=1),color='red') +
  labs(title='Height in feet') +
  aes(x=Age,y=height) +
  stat_summary(fun.y='mean',geom='point') +
  geom_smooth()

ggplot(crews %>% filter(height>36,height<7*12),aes(y=height/12,x=Age)) + labs(title='Height in feet') +
  scale_fill_gradientn("Number of sailors",trans='log10',colours=heat.colors(10))


crews %>% filter(height>36,height<7*12) %>% ggplot() + aes(y=height/12,x=as.Date(date))+ geom_point(siz


ggplot(crews %>% filter(!is.na(Hair)))+ geom_bar(aes(x=Hair)) + coord_flip()
```

Text Scatterplots

For humanities data analysis, text scatterplots are particularly important.

For them, you can use the `geom_text` function. It works just like `geom_point`, but requires an additional aesthetic: `label`.

What does this chart show? What variables (if any) do you think are driving this chart?

```
crews %>% group_by(Residence) %>% mutate(count=n()) %>% filter(count>50) %>% summarize(averageHeight = m
```