Topic: Data Visualization Feb 11

Erich Denk

February 11, 2019

There is such a thing as a bad data visualization! We do get "cognitive amplification". We can more easily pull information out.

There are some common mistakes. Scale can matter a great deal to the story being told. Also, consider what kind of distribution you might be dealing with

Our meaining in life is to make plots that don't even need a caption (though still add a caption). We want to be able to see when plots are terrible and misleading too.

Color

Color can be helpful in a few ways. - Color is a great tool to distinguish across - It can also represent a range of values. - It can highlight important/unexpected values.

Presentation as Information v. Distortion

Exploratory data analysis can be fruitful in finding research questions. But be careful that it doesn't show things that don't exist. For example a density plot that suggests that data exists where it doesn't. Or perhaps bin width in a histogram.

The type of data shoudl drive the visualization decsisions that you make. Type disctates the scale and consequently the best visualizations for your data.

The Grammar of Graphics

There are underlying principles that plotting mechanisms should have. ggplot2 uses the language that allows you to create sophisticated graphics with ease.

There are four underlying components: 1. a dataset - wrangled and *tidy* 2. coordinate system - easily comes out of the box 3. mappings - the varaibels we want to visualize 4. **geom** etric expressions of how to project them onto space

Let's look at them using the built in diamond grou

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```r
glimpse(diamonds)
```

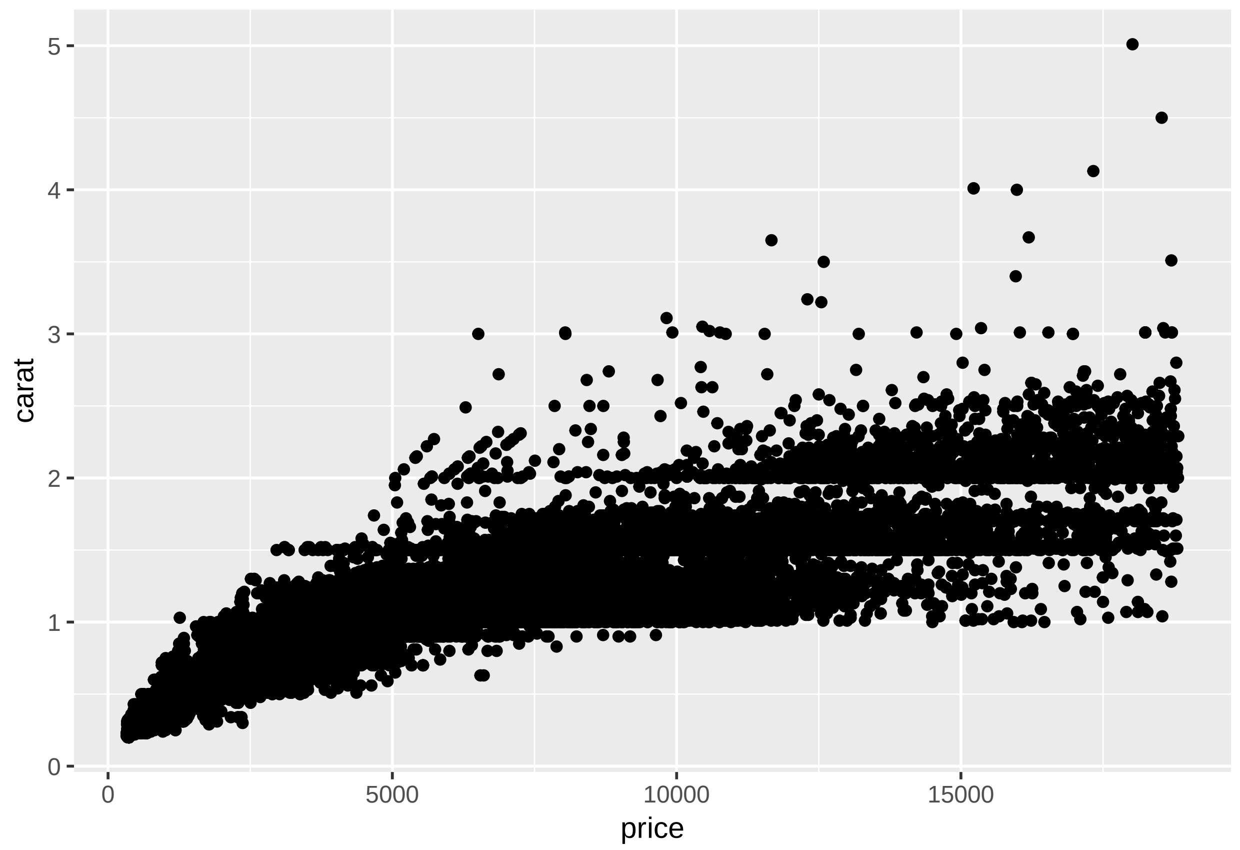
```
## Observations: 53,940
## Variables: 10
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, ...
## $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very G...
## $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, ...
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI...
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, ...
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54...
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339,...
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, ...
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, ...
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, ...
```

The big trick is the mapping. What variable makes up the y axis and x axis? Do we want to group any. You *must* use the `aes()` function to wrap it.

The `geom` points are just layers where you can add them together. How do we want to display the data. We can add a few geoms together. Like adding another coat of paint to the process.

An extremely helpful ggplot function is that we can assign a plot as an object and render them later. Save the data object and reload the data object.

```r
ggplot(data = diamonds, aes(x= price, y = carat))+
  geom_point()
```



Here are the things we want to think about. We can add as many geom layers as we want. Just so that we use the +. Then we can get really sophisticated and use `scale_function` `theme` functions, `facet` functions.

If it comes from the data, it needs to be wrapped in the `aes()` function. A common mistake. As always check out the beautiful data visualization cheat sheet.

`ggsave` is also particularly helpful. You can render as a pdf, a png if you are trying to put it on the internet. Don't pass over "eps", "ps" and "svg" vector style images. They are useful in Adobe Illustrator.

Example 1: Regime Type and Economic Development

Some helpful additions `geom_jitter` adds a little noise so that we can break points around. Useful when you have points in a categorical scale.

`alpha` is an option that adjusts transparency. Helpful for showing density

We can simply change the geom projection to `geom_boxplot`

`theme_economist` exists, which is very cool

`facet_wrap` the ~ "this many types" of plot. Actually like a function/equation. `free_x` is a fun option to consider.

`fill` is anything you would have to shape in. You can say fill by a varaible, which produces a legend on the side.

`theme()` can be a great catch all. Allows adjustment of legends, their positions, etc. Kind of a twisting of dials. You can change position via option`hjust`

`scale_AESTHETIC_manual` allow customization of a number of things.

`labs` allows us to specify lots of features. subtitles, types of text, positions.etc.

Example 2: Process of Democratization in South America

Now we have subset the data to only South American countries.

`unique` shows a vector of unique values for a given variable.

Using facet wrap is best when you have too many types to have on one plot. Yu can also specify the number of columns.

`ggthemes` is another package outside of the ggplot feature. Can call it via `ggthemes::` and do like fivethirtyeight's stylebook. If you use ggthemes you might need to adjust inside the theme function to override what has been presented via the ggtheme adjustments.

Example 3: Visualizing the Democratic Wave in South America

We want to show that all these things kind of happened at the same time! We can add a third layer to this all.

`geom_tile` places tiles for each spot. But we can fill via a variable. Just need to wrap in `aes` warpper function.

`coord_flip()` flips your coordinates.

`scale_fill_gradient2` allows you to pick three colors. An end, middle and start point.

You don't want gridlines in a tile plot. `ggthemes:theme_tufte` does this. After the guy who blocks people on twitter who disagree with him on plotting.

Always think about both medium and audience.

Dates

They can definitely be a nuisance. We have to deal with them on R's terms.

They are treated as a string in R. We need to transform it into a date structure in the same way that we coerce a dataframe or factor or whatever. We need to tell R the format

Can be helpful in finding time differences too.

```r
example <- "February 3, 1987"
as.Date(example, format = "%B %d, %Y" )
```

```
## [1] "1987-02-03"
```

See a table for how. Make sure you keep the punctuation consistent from how date is presented. Could be all smushed together.

`lubridate` a tidyverse like package that you can use to make things a lot easier.

You can also use lubridate for more complex functions. Easier to use than to look up the syntax of how to specfy each date.

```r
require(lubridate)
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```r
mdy(example)
```

```
## [1] "1987-02-03"
```

Rounding dates can be done very easily. Just option `unit = "week"` or whatever you need.