GGPLOT: Creating Graphics Logically

Amit Gal

Graphics in base R

- Easy to create, and they look ok.
  - Class sensitive commands
- Everything can be adjusted through parameters
  - But sometimes hard to find the right ones
  - Require lots of trial-error, no easy patterns.
- Graphs are not R objects
  - Cumbersome handling of “devices”
  - Hard to manage, save, update, reproduce...

Meet GGPLOT

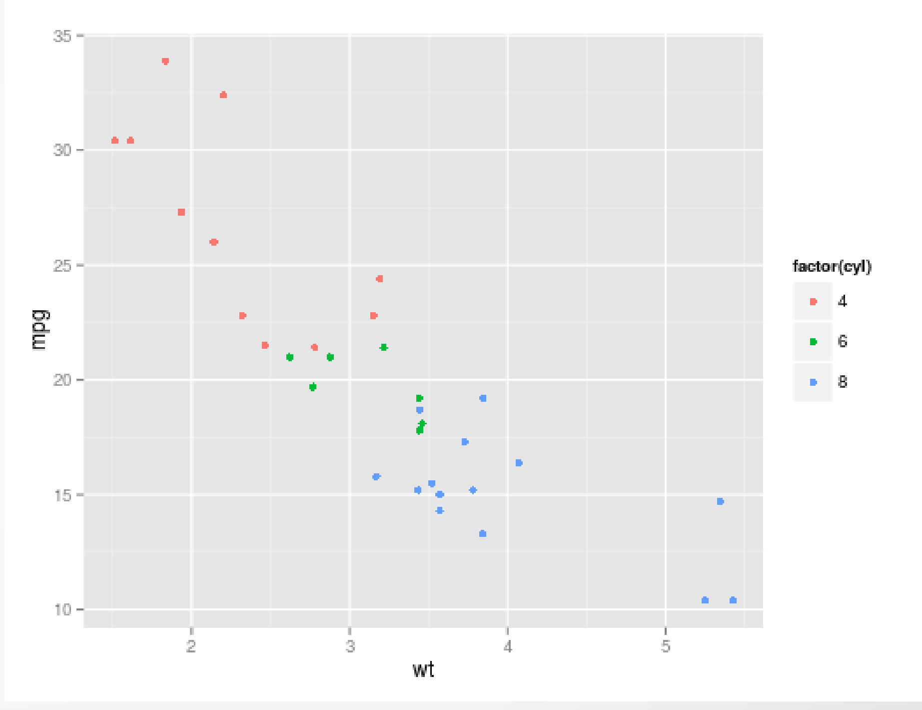
- The promise of a “grammar for graphics”
- graphs are standard R objects
  - Easy to manage, save, etc.
- Graphs are composed of layers
  - Easy to add stuff to existing graphs
- Provides a separation between content and visualization strategy
  - Enables creation of reproducible visualization patterns.

Why grammar?

- Grammars are generative systems – they provide means (e.g. rules, patterns, etc.) for constructing meaningful graphics from building blocks
- If you think of a graphic as a message (and you should!), grammars are way to distinguish between well formed “sentences” and those that are not so well formed.
  - Also, grammar as a syntax serves as a basis for a semantic system. After all, you want your graphics to be meaningful!
- Abstraction of the visualization process
  - How we get from data to graphs
- WYSIWYG vs. WYMIWYG
  - A useful metaphor: Latex vs. Word for creating scientific documents

Basic elements of the grammar

- “A graph is a mapping from data-space to visual-space” (Hadley Wickham)
- An example: a simple scatterplot
  - Wt is mapped to the x axis
  - Mpg is mapped to the y axis
  - Cyl is mapped to the color property



Basic elements: The aesthetics

- In its most generic terms a graph consists of some geometric features (e.g. points, lines, text labels, etc.) in some 2-d space (the screen or paper)
- Each geometric feature has some properties that it must know in order to put it “in the right place”
  - For a point, you need its x,y position + color + size
  - For a bar, you must know its bottom-left position (x,y), its height, width, color, fill, etc.
- Each such required property for a geometric feature is called “aesthetic”. It is an abstract template of the data required to create the plot
- The first step in generating a plot, is mapping variables from the data, to the various aesthetics required. This is done using the aes() function, and passed to ggplot through the mapping parameter:

  ```
  > ggplot(...,mapping = aes(x=wt,y=mpg,color=factor(cyl)) ...)
  ```

Basic elements: Scales

- The aes thetic mapping establish the “source” and the “target” domains of the mapping. We need two more elements: scales and statistics
- The original data is given in “data units” (for example the “cyl” field is 4 6 or 8. but the graphics may need different units (e.g. “red”, “blue”, “green”).
- Scales define how we measure the aesthetics. We must have a scale for each aesthetic element. e.g. there must be a color_scale if we use the “color” aesthetic, and so also an x_scale and y_scale, etc.

Basic elements: statistics

- Statistics is the function that actually convert the data to aesthetics.
- Sometimes it is an easy transformation:
  - In scatterplot, typically the identity function is used for x and y, the “color” is a simple 1-1 map from the levels of the given variable to the list of colors.
- Sometimes it is more difficult. In fact the “statistics” part takes the data as input and create a new data.frame as output. This new data.frame must have all the required values for the aesthetics, scaled correctly.
- let's explore how it is done for a histogram:

Creating a histogram

- The geometric object is bar and it needs its x location and height and width set.
- The height should match the count for a given range, but we don't have it in the data, so we must compute it.
- We use the “bin” statistics to partition the values of the variable mapped to x into bins, and create the ..count.. variable and width. The internal results should look like:

The histogram data dynamics

X
1
1
2
3
5
6
5
1
4

“bin stat”

x	width	..count..
1	2	4
3	2	2
5	2	3

mapping

We can now create a layer!

- ggplot organizes graphs in geometric layers. Each layer is dedicated to a single type of geomtry, with data, an aesthetic mapping and a statistics for relevant data transformation. Here is a typical code:

  ```
  df <- data.frame(x = c(1,1,2,3,5,6,5,1,4))
  ggplot() + layer(
          data = df, mapping = aes(x=x,y=..count..),
          stat = “bin”, binwidth = 2, geom = “bar”) +
          scale_x_continuous() + scale_y_continuous()
  ```

- Layers can be stacked, but for a plot to work some consistency must be kept:
  - Key data must be kept constant across layers
  - No mapping of to different variables to the same aesthetics
  - Typically no change of scales is allowed between layers, but some transformation are possible

We are almost done...

- Coordinate systems
  - This is one last bit that enables to map the scales we use (which are abstract) to actual physical location (on the screen.
  - Typically we use cartesian coordinates by adding the:

    + coord_cartesian() to the expression. But other possibilities exist (e.g. polar coordinates to create, for example, pie charts and radar-like plots)
- Defaults
  - Most of the parameters need not be explicitly stated.
    - For example, if you used “bin” statistics, then by default it also uses the “bar” geometry.
  - Default values are grouped together to create styles or themes that create a sense of unity between your graphs. You can change these by simply adding a theme declaration to your graph

    > my.graph+theme_bw()

Shorthand style

- Because of the many defaults and commonalities between layers, one can use shorthand style to define graphs:

  ```
  ggplot(df,aes(x=x))+geom_histogram(binwidth=2)
  ```

- There are many “geom_XXXX” functions that pre-define layers.
- There are also other “helper” functions to control scales, coordinates and other properties (such as ggtitle for adding title, etc.)
- There is an even shorter version to create ggplots - using qplot()
  - qplot() makes stronger assumptions (e.g. make guesses about your intention) thus enabling defining a graph with even less hassle, regaining some of the original plot() function flexibility

Summary of the basic principles

- A plot is a map between data and visual elements
- It consists of layers that share some common properties
- Each layer has: data, aesthetic mapping, scales, geometry, a statistical transformation and a coordinate system – together they define how the plot will look like.
- Layers and some additional properties can be added on top of any graph – as long as there are no contradictions between the definitions
- We are now ready to look deeper into some graphs and “reverse engineer” them to gain some deeper insight into the power of ggplot. So let's open R-studio and the script I prepared.