Dataviz packages: a travelogue

Ben Klemens

28 August 2016

I've toyed with base R, ggplot2, Gnuplot, matplotlib, Bokeh, D3, some D3 add-ons I've already forgotten about, some packages I tried for five minutes and walked away from. This is a post by a tourist about the problems these packages face and how they try to solve them, and why being a tourist like this is such hard going. Next episode will be entitled 'D3: a travelogue', because I think D3 is worth extra attention. Keeping to my character, this series will be largely conceptual and have no visuals.

The short version of the plotting problem: so many elements, each of which needs to be tweaked. On the basic scatterplot, you've got axis labels, major and minor tic marks, scale (linear? log?), a grid (if any), the points themselves and their shape, size, positions, color, transparency, a key or legend and its content. And a title.

Meanwhile, everybody wants to be able to type `plot(height, weight)` and get a plot. A system that doesn't have that capability of one-command plot generation would have such a high hurdle to adoption that it would need regular exposure on the cover of the New York Times to get any traction.

The three-layer model I think of all of these dataviz packages as three-layer systems:

Bottom layer In the end, we have to get dots on the canvas, be it a Mac/X/Windows/browser window, a png, a gif, an SVG, a LaTeX diagram, or ideally all of the above. As with any effort toward device-independence, every package author works out a set of primitives to draw lines, paint dots, write text sideways. Sometimes they let you see them.

Middle layer Step past abstract drawing and manipulate elements we commonly accept as part of a plot: axes, points, histogram bars.

Top layer Build the entire picture with one command: `plot(height, weight)`.

The SDVW Let the *standard data viz workflow* (SDVW) consist of starting with the one-line command from the top layer to generate a plot, then applying a series of middle-layer commands to adjust the defaults to something more appropriate to the situation.

I really do think this is the standard. It is generally how I see people talk about a plot ('It's basically OK, but this one thing looks off'). It is easy to understand and implement: get the basic plot up, then tick through every element that needs modification, one by one. What could go wrong?

The most common other workflow may be the exploratory workflow: add a top layer command to a script or call one from the command line, try to learn something from the plot displayed, then throw it away. When working to a tight deadline for people who think Excel charts look good, the workflow may be to call the top layer command and let all of the defaults stand.

Ease of (initial) use *Ease of initial use* is when the top-layer command quickly produces something attractive out of the box, and is what is demanded by the exploratory workflow and the under-a-deadline workflow. It is also often what is demanded by people new to the package, who want to see fast positive feedback.

Ease of use is about the middle layer. No matter how stellar the defaults, your are plotting unique data, so it is a probability zero event that the defaults are exactly correct. For the same reason, the SDVW involves slightly different changes every time.

There's a neat mapping between the SDVW and ease of (initial) use: is the focus on making the top layer or the middle layer fast and easy? In an ideal world, they would both be, but our time on Earth is finite, and package authors have to lean toward one or the other.

To give an example of a system that prefers ease of use over ease of initial use and thus maps nicely to the SDVW, we have that unfashionable workhorse Gnuplot. At the top layer, start with a `plot` or `splot` command to generate the plot. At the middle layer, every component either has a `set` command which you can find in the awkward interactive help system, or can't be modified so stop trying. If you want to layer components, like a regression line through a scatterplot or a Joy Division plot[1], use `replot`. So the inital plot is one command, everybody thinks the defaults are ugly or broken, but there is a straightforward mapping between changes you may want to make and `set` commands to implement them.

Non-orthogonality Although our ideal in most software engineering is to have orthogonal components where each tweak stands independent of every other, a plot is unified. Text labels should be close to the points they are marking, but *close* is defined by the context of point density, space on the page, text font. As a user, I don't even want orthogonality: if I expand the text font, I want the definition of *close* to self-update so nothing gets cut off.

The smarter the top layer gets, the more difficult it is to get the middle layer to behave straightforwardly, and the less likely it is that there's a Gnuplot-like `set` command that just makes one tweak. For example, in some systems, adding a second y-axis is basically orthogonal to everything else and so is trivial; in ggplot2 axes are more closely tied to everything so adding a second y-axis is a pain.

Documentation by example Documentation in this genre leans toward examples of the SDVW. Here are some bl.ocks[2], now cut/paste/modify them to what you want. I found this example-driven form to be amazingly consistent across packages and authors in this space. It's not just the official documentation and blog entries: I have access to

[1] `http://www.gnuplotting.org/creating-pseudo-3d-plots/`
[2] `https://bl.ocks.org`

Safari Books[3] [self-promotion disclaimer: every time you read a page from *21st Century C* there, they pay me a fraction of a cent.], where I went hoping to find full expositions going beyond worked SDVW examples, but I instead found more detailed and extensive SDVWs. I'd check a question at Stack Overflow, and the answer is a complete example preceded by 'here, try this.' This is in contrast to other times I've gone to Stack Overflow and get the usual paragraphs of mansplaining about a single line of code.

Example-driven documentation is a corollary to the lack of orthogonality. How does it make sense to modify the points in a scatterplot when you haven't even produced the plot? But because each SDVW is different, either the example did what you are trying to do and you've won, or it isn't and you are right where you started.

Every package provides examples of the top three or four things you can do to modify the axes, but surprisingly few take the time to provide a boring page listing all the things you can do with the axes and how, or the page is sparse and descriptions point you back to the examples. Sometimes the answer on Stack Overflow would be a one-setting tweak which is never mentioned in the official documentation. I'm harping on documentation because I found it to be the biggest indicator of whether the authors were shooting for ease of initial use or ease of use, and on a cultural note it shows a clear difference between how dataviz package authors understand their users and how general data analysis package authors see their users.

That concludes part one of my travelogue. These systems do amazing things, but this is my confession that the full SDVW is still a slog to me. The standard data viz workflow is, to the best of my understanding, standard, yet being a tourist across packages required adapting to a new idiosyncratic way of walking through the SDVW every time. This may be because of the vagaries of how different base layers were designed, the (possibly hubris-driven) sales pitch that all you need is a top-layer command and you'll never need to change anything else, or the fundamental non-orthogonality of a data visualization.

[3] `https://safaribooksonline.com`