D3: a travelogue

Ben Klemens

29 August 2016

Here's my largest project in D3: a visual calculator for your U.S. 2015 taxes[1]. It doesn't look much like a scatterplot, but it uses a tool, Data Driven Documents (D3), used extensively for traditional dots-on-a-grid plots. I'm going to discuss the structure of this system, in the context of the last entry (entry #195), which went over the parts of the standard data viz workflow (SDVW). If you've never used D3 and the acronym *DOM* doesn't have meaning to you, reading this may soften the initial blow of trying to use it.

The DOM Document Object Model. The document on your web page screen is a parent object with a well-defined set of child objects, akin to standard programming language structs that hold other structs, or XML documents whose elements hold other XML elements. Each struct holds elements of any sort: scalars, arrays, sub-objects, functions. Being a simple implementation of the struct with sub-structs, the DOM as a whole is a work of clean generality.

The DOM is a work of massive hyper-specificity, as each type of object—header, canvas, Scalable Vector Graphic, rectangle, text, whatever—has associated its own set of special properties which your browser or other reader will use to render or act upon the object. Your browser's developer tools will show you the full tree and associated properties. Further, there is no fixed list of what those properties are. If you want to add `falafelness`, and then asssign a javascript function to the object's `deepFry` property to modify the object's CSS styles based on its falafelness, all that is entirely valid. This right to assign arbitrary magic words is also open to the author of any library, D3 included.

Further, there are the quirks of history, as these neat objects represent web pages which include different HTML markers like IDs and spans, which tie in to cascading style sheets (CSS). Add it all together and any one object has attributes, tags, styles, properties, content, events. Changing an object is almost always a straightforward tweak—once you win the seek-and-find of determining which attribute, tag, style, property, content, or event to modify.

The HTML Histogram Stepping back from the potential for massive complexity, here's is a simple demo of a horizontal-bar histogram. The table has four rows, each with an object of class `bar`, where the style characteristics of any `bar` object are listed

[1] `https://b-k.github.io/1040.js`

in the header. Then in the table, the `width` style is set for each individual bar, like `style="width:200px;"`.

```
<html>
<head>&lt;style&gt;
.bar {height: 10px; border: 2px solid; color: #2E9AFE;}
</style>&lt;/head&gt;

<body>
<table>
<tr>&lt;td&gt;Joe&lt;/td&gt; &lt;td&gt;&lt;div class=”bar” style=”width: 80px;”/&gt;&lt;/td
    &gt;&lt;/tr&gt;
<tr>&lt;td&gt;Jane&lt;/td&gt; &lt;td&gt;&lt;div class=”bar” style=”width:300px;”/&gt;&lt;/
    td&gt;&lt;/tr&gt;
<tr>&lt;td&gt;Jerome&lt;/td&gt; &lt;td&gt;&lt;div class=”bar” style=”width:300px;”/&gt;&
    lt;/td&gt;&lt;/tr&gt;
<tr>&lt;td&gt;Janet&lt;/td&gt; &lt;td&gt;&lt;div class=”bar” style=”width:126px;”/&gt;&lt;/
    td&gt;&lt;/tr&gt;
</table>
</body>
</html>
```

Using any programming language at all, you could write a loop to produce a row of this table for each observation in your data set (plus the requisite HTML header and footer). Using only HTML and CSS, you've generated an OK data visualization.

You've probably already started on the SDVW in your head, and are thinking that the spacing is too big, or the shade of blue is boring, or the label fonts are wrong. And, of course, everything you would need to make those changes is in the DOM somewhere. Want the bars to go up instead of rightward? Set the `height` style on a per-bar basis instead of the `width` (and rearrange the table...). Want to give the girls pink bars? Neither I nor the HTML standard can stop you from setting `color` conditional on another column in the data set.

The Grammar of Graphics The GoG is a book by Leland Wilkinson, subsequently implemented as various pieces of software, the most popular of which seems to be `ggplot2` for R. I have no idea whether the HTML histogram or the GoG came first, but their core concept is the same: the objects on the screen—one per observation—have a set of characteristics (herein *æsthetics*), and we should be able to vary any of them based on the data. For example, maybe box height represents an observed value and color represents a statistical confidence measure.

Gnuplot and earlier plotting programs don't think of plots as objects with æsthetics. The æsthetic built in to the top-level plot command is (X, Y, Z) position, and others can be linked to data if there is a middle-layer function that was written to do so. ggplot2 is much more flexible, and every(?) æsthetic is set via the top-level command, but each geometry still has a fixed list of æsthetics. After all, if you want to do something unusual like change the axes' thickness based on data, somebody had to write that up using R's base-layer graphics capabilities.

Applying the GoG principle, setting object æsthetics using data, to the object properties of the DOM is natural to the point of being obvious, as per the HTML histogram. D3 just streamlines the process. You provide a data set, and it generates the right number of objects in a scatterplot, or bars, or graph, and applies your æsthetic rules to each. It provides an event loop so that the points can be redrawn on demand, so a button can switch how the drawing is done, censor some points and uncensor others, or update on a modified data set. The workflow starts with a set of top-layer commands corresponding to plot types, with ring plots, certain network plots, tried-and-true bar charts and scatterplots. To make the tax graph, I used dagre-d3[2], a top layer to draw directed graphs.

So we can get a lot done at the top layer in a GoG-type implementation, because a lot of the little tweaks we want to make turn out to fit this concept of applying data to an æsthetic. For those that don't, we have the ability to modify every property, attribute, tag, and style. I'm not sure if any custom-written base layer of any data viz package will ever be able to achieve that sort of generality, and the SDVW via D3 feels accordingly different from the SDVW in a system with a fixed set of middle-layer commands to tweak a plot.

As a tourist, the difficulties I had with D3 were primarily about getting to know the DOM and its many idiosyncracies. The documentation clearly expects that you already know how to manipulate objects and that the workings of the HTML histogram is more-or-less obvious to you. But this is also D3's strength. I had a mini-rant last time about the documentation of dataviz packages, which are often lacking in the item-by-item specifics one needs to make item-by-item changes to a plot. Meanwhile, web developers do nothing better than write web pages documenting web page elements, so the problem is not about finding hidden information but managing the sense of being overwhelmed. You are empowered to make exactly the visualization you want.

[2] `https://github.com/cpettitt/dagre-d3`