Getting Started with D3 for Visualising Web Data

Introduction

This tutorial is based on Max van Kleek's excellent tutorial for the Open Data Institute Summit 2014. You can find the original tutorial at https://github.com/sociam/WS3-D3-Tutorial

D3 is a data-oriented Web micro-framework that allows Web programmers to focus on data rather than everything around it. It follows the tradition of jQuery to provide 'fluent interfaces' that makes code extremely succinct and (sometimes!) easy to read.

Unlike using Matlab, R, SPSS or Excel, however, it is a general-purpose data drawing API. This means two things: first, that it takes a bit more time and investment to build things in D3 than it does to plot things in any high level tool or statistical analysis toolkit. But it also means that you have far greater flexibility about what you can build - in fact, you can probably build almost any kind of data visualisation you can think of with the right amount of D3.

NOTE: ↵ means there is a line break for print purposes only – the code on the next line should continue on without any spaces!

Getting Set Up

This tutorial assumes a familiarity with Web technologies used to build Web sites. JavaScript knowledge is recommended but not necessary – if you decide to explore D3 further it will be required, however. Max recommends "Eloquent JavaScript" (http://eloquentjavascript.net/ contents.html) for beginners, and "JavaScript: The Good Parts" for advanced JS programmers.

Setting up your environment

- Install Node.js, which includes the npm package manager: https://nodejs.org/en/
- Then create an empty directory (folder) where you would like to run the tutorial
- Navigate to the directory in a terminal / command line
    - (Mac users can right-click the folder in finder and use the Service menu to open a Terminal window at that folder.)
    - Run the following to set up a static web server and a manager for the dependencies we need to build a web site.
        - **`npm install -g http-server bower`**
          Note: on OS X/Linux, you will need to add 'sudo' to the beginning of that command
        - **`bower install jQuery d3 underscore`**
            - This command is *case-sensitive*. Type exactly as above.
            - This is installing local copies of the libraries you need, rather than accessing their live online version.
- Now you need some data to play with. Twitter data can be harvested using the University's tweet harvester: http://tweets.soton.ac.uk (you need to be on the Southampton network or connected to the VPN to access this site). You can then export the results as json files, which is what we'll be using for this tutorial. Navigate to the file location given in the class, and download **`'tweets.json'`**
    - Alternatively, if you really want to collect your own Twitter data, follow Step 1b in Max's instructions at https://github.com/sociam/WS3-D3-Tutorial/blob/master/exercises.pdf
- You should now have an environment set up, and some data to use. Now we just need some code. Note – all of the code is available from Max's github. If you get stuck, have a look at the code there.

Prepare your Code

First, make a very basic html file in your favourite Text Editor. I use TextWrangler on Mac. Call it '**index.html**', with the following contents:

```
<!doctype html>
	<html lang="en">
	<head>
		<link href="index.css" rel="stylesheet"/>
		<title>Playing with D3</title>
	</head>
	<body>
		<script src="bower_components/jquery/dist/jquery.min.js"
	type="text/javascript"></script>
		<script src="bower_components/underscore/underscore.js"
	type="text/javascript"></script>
		<script src="bower_components/d3/d3.min.js"
	type="text/javascript"></script>
		<script src="index.js" type="text/javascript"></script>
	</body>
</html>
```

Note that we have included references to the bower_components directory where our dependencies are installed. Check that the paths are correct.

Now we need a CSS file, '**index.css**':

```
* {
	box-sizing:border-box;
	font-family:'helvetica-neue';
	font-weight:100;
}

body {
	padding:10px;
	height:100%;
}

svg {
	width:100%;
	height:600px;
	border:1px solid #eee;
	box-shadow: 5px 5px 5px #eee;
}
```

What do you notice about this? Answer on the next page!

We have CSS styling for an SVG object. But, that object isn't in our HTML file. The SVG object will be created using D3 and then used as a space to draw our data.

Now finally, we need a JavaScript file to access the data and do something with it. Create an empty file **'index.js'**, and then put into it:

```
jQuery(document).ready(function() {
	var svg;
	var setup = function() { svg = d3.select('body').append('svg');  };
	$.get('/tweets.json').then(function(tweets) {
		console.log('tweets > ', tweets.tweets);
		setup();
		window.tweets = tweets.tweets;
	}).fail(function(data) {
		console.error('error loading data ', data);
	});
});
```

See the 'svg = d3.select('body').append('svg');' line? We're creating a new html element in the body of the html document. This is our canvas on which to work!

This code uses JQuery to access the tweets file (ensure it is in the same folder as your code). It then prints them out to the console.

Checking it works

Remember we set up a local web server earlier? Now we need to start it.
In your terminal, type

`http-server`

If you installed it correctly, you should see something printed out about an http-server starting up. Now, don't close down the terminal until you're finished with the tutorial. If you need to carry out any more commands, you should be able to open a new command prompt / terminal window or tab.

Now, go into your preferred browser, Chrome is a good option for this, and navigate to

localhost:8080

You should have a blank page. Fantastic! If not, something has probably gone wrong along the way, go back and check you've followed the steps above correctly.
To check something has worked behind the scenes, open the browser console (On Chrome: View->Developer->Javascript Console) and you should find a message with something like:

Tweets > [Object , Object , Object ……..]

You can click on any of these Objects to see some details about the tweet they represent.

Plotting the data

Now, we need to establish what we want to measure about the tweets. We need to compute X and Y axes values to compare in a scatterplot.

Let's look at whether the number of words in a Tweet correlates with the number of retweets it receives. Obviously, it shouldn't, but let's look at it to show this anyway.

To achieve this, we need to modify our index.js file a bit.
Firstly, modify the second line so that we now have:

(1)
```
var svg, hmargin = 40, vmargin = 40, xscale, yscale, width, height;
```

This declares some variables that we'll need later.
Then for the next line, add **var** to the beginning, as we will need to add in some code before this.

Before
```
var setup = function() { svg = d3.select('body').append('svg');  };
```
we need a function for the x axis that returns the number of words in each tweets:

(2)
```
var xval = function(tweet) {
      return tweet.text.split(' ').filter(function(x) { return x.trim() ↵
.length > 0; }).length;
};
```

and another that returns the retweet count of that particular tweet for the y value:

(3)
```
var yval = function(tweet) { return tweet.retweet_count; };
```

Now, we need to create a plotting function, to handle the drawing of a scatterplot. The variables we added earlier are needed now.

We need a **plot** function (full code listing below, but read the explanation first)

The function we'll write is divided into three sections. The first, **scaling**, serves the purpose of declaring D3 scales, which are functions responsible for transforming the raw values we are plotting into SVG coordinates. Since we are plotting two different ranges of values against each other, we need two separate scale functions: one for x (the number of tokens) and one for y (the number of retweets). For each scale, we need to tell them explicitly what the min and max values of the data we will be plotting (the 'domain') is, and the output coordinates in SVG space we desire (the 'range'). See the D3 documentation on d3.scale for the complete scale API.

The next, **drawing** section is where the party goes on; this simple "single line" function processes and then plots the scatterplot's dataset. This is where D3 really shines. The way to read this is as follows:

1. First, select (in a jQuery sense) all circle elements with class `pt` in the SVG canvas, and data join them against the array tweets. This means, pair up each element in the selection, one at a time, against the elements of the data array, in order. The question you might ask yourself is – what if there are no elements yet to select? We haven't constructed any circles! This is where D3 and jQuery differ. D3 takes note of everything that doesn't exist yet and puts it into what it calls an enter selection. To get access to the enter selection, call the `enter()` method, which is what we do next. Data-binding and enter/exit selections are fundamental to D3, and so please consult the documentation.

2. For those elements in **`tweets`** that don't yet have a corresponding circle.pt element, create a new circle element and **`append()`** it.
3. Now, for every **`tweet`** object circle, set the '**`cx`**' ("centre-x") attribute of the SVG circle to the value returned by **`function(t) { return xscale(xval(t)); }`** for each tweet provided; note that xscale is the scale value we declared earlier and xval is the function that measures the number of tokens in each tweet. Similarly, set 'cy' (centre y) to the value to **`function(t) { return yscale(yval(t)); }`**
4. Set each circle's radius to 3, and its class to 'pt.'

The final section pertains to drawing the axes, using D3's axis convenience methods. The axes directly read the **scale** variables to identify where and how to draw the extrema. Since the x-axis needs to be at the bottom of the display, we use an SVG transform **translate** operation for that.

```
var plot = function(tweets) {
            width = $('svg').width();
            height = $('svg').height();

            xscale = d3.scale.linear().range ↵
([hmargin, width-2*hmargin]);
            yscale = d3.scale.linear().range ↵
([height-vmargin,vmargin]);

            var xvals = tweets.map(xval);
            var yvals = tweets.map(yval);

            // establish the domain here.
            xscale.domain([_.min(xvals), _.max(xvals)]);
            yscale.domain([_.min(yvals), _.max(yvals)]);

            svg.selectAll('circle.pt').data(tweets)
              .enter().append('circle')
              .attr('cx', function(t) { return xscale(xval(t)); })
              .attr('cy', function(t) { return yscale(yval(t)); })
              .attr('r', 3)
              .attr('class', 'pt');

            // add axes:
            var xaxis = d3.svg.axis().scale(xscale).orient('bottom'),
                  yaxis = d3.svg.axis().scale(yscale).orient('left');
            svg.append('g')
                  .attr('class','xaxis')
                  .attr('transform', 'translate ↵
(0,'+(height-vmargin)+')')
                  .call(xaxis);
            svg.append('g')
                  .attr('class','yaxis')
                  .attr('transform','translate('+3.0/4*vmargin+',0)')
               .call(yaxis);
      };
```

Scaling

Drawing

Draw and add axes

Add this code after the line **`var = yval …..`** and before you declare the **`setup`** variable.
You then need to make the following additions to the final piece of code (additions highlighted in bold):

```
$.get('/tweets.json').then(function(tweets) {
            console.log('tweets > ', tweets.tweets);
            data = tweets.tweets;
            setup();
            plot(data);
            window.tweets = tweets.tweets;
      }).fail(function(data) {
            console.error('error loading data ', data);
      });
```

The first of these lines stores the contents of the 'tweets' array from the json file in a variable. Open the .json file and have a look. It has one root element, the 'tweets' item, which contains all the various tweet details. The second line simply calls the plot function on this data, passing over the tweets to be visualised.

Ok, you should be ready. Refresh your browser and you should see a scatter plot. Check for syntax errors if anything isn't working.

Change to log scale on the y-axis

You will notice that some tweets got many retweets while the majority got very few. We can make this more readable by viewing a log scale for the y-axis.
Simply change the line for the yscale from **`d3.scale.linear()`** to **`d3.scale.log()`**

And then for yval, change the return statement that you wrote earlier (code block 4) from
`var yval = function(tweet) { return tweet.retweet_count; };`
to:
`var yval = function(tweet) { return tweet.retweet_count + 1; };`

Referesh the page again, and things should be a bit easier to see now. Now, you may want to make the log scale on the y axis a bit friendlier, and replace the values on it with their original numeric values. For this we can modify the line where we add in the yaxis by using D3 chaining of commands. So from:

```
var yaxis = d3.svg.axis().scale(yscale).orient('left');
```

You can simply add additional code to the statement to further customise the 'tick format' of the scale:

```
var yaxis = d3.svg.axis().scale(yscale).orient('left').tickFormat(↵
function (d) {
      return yscale.tickFormat(4,d3.format(",d"))(d)
});;
```

Adding Interactivity

It would be nice if we could see which tweets correspond to each point on the plot. Let's use mouseover-display to show us the content of each tweet.

Within index.html, we need to add something to display the tweet contents. Within the **`body`** tag add the following code, before you call the various scripts. This creates the divs ready for where we'll display the tweet information.

```
<div class="tweet-display">
      <div class="author"></div>
      <div class="text"></div>
</div>
```

Now, within index.js, we need to add something to handle mouseover events on the plot, which will populate those divs with content.

Add in the following, before the '**// add axes:'** line.

```
$('circle.pt').on('mouseover', function(evt) {
      var target = evt.target;
      $('.tweet-display .text').html(target.__data__.text);
      $('.tweet-display .author').html("@"+target.__data__.user.screen_name);
      $(target).attr('class', 'pt selected');
});
$('circle.pt').on('mouseout', function(evt) {
      var target = evt.target;
      $('.tweet-display .text, .tweet-display .author').html('');
      $(target).attr('class', 'pt');
});
```

The two event handlers simply watch for hover events over any of the points. The hovered-over DOM object (circle) is available as **target** on the first argument, which we named **evt**. To get access to the actual data object that D3 has joined to it, we use the **.__data__** field on the DOM object; which gives us access to the raw tweet that we joined against the object.

Add in the following css to style your new creations:

```
circle.pt {
      fill:red;
      stroke:none;
}
circle.pt.selected {
      stroke-width:1px;
      stroke:#2ff;
      fill:white;
}
.tweet-display {
      position:absolute;
      left:20px; right:20px;
      top:10px;
      font-size:12pt;
}
.tweet-display .author { color: #2ef; }
.tweet-display * { display: inline-block; }
```

This provides style information for your circles so that the selected one looks different to the rest. And it provides style information for the tweet-display div that you created earlier, so that the author of the tweet is highlighted.

Now refresh your page. If everything has gone correctly, you should now see the content of tweets when you hover over them on the plot.

Moving On

Now, take a look at some D3 examples and if you want, add in more interactivity. Alternatively, move on to Part 2 of Max's tutorial at the following link:

http://bit.ly/1SVD67g

This will guide you through adapting an existing example to pull in the data file we've been using.

Another tutorial to try out if you want to learn more is at:
https://www.dashingd3js.com/table-of-contents