D3 Lab 1

Today we begin working with D3 in class. Work through as much as you can in class, and finish on your own time. Some optional D3 resources are listed at the end of the file. Submit by Wednesday Jan 12 at 10pm.

What to post to Canvas: Zip your HTML/JS files and name the zipped folder with your name. Name each individual HTML/JS file with which part it corresponds to (e.g., part5.html and part5.js).

Thanks to Jessica Hullman for the original version of this lab.

Part 1

To create visualizations we map data onto visual channels. The most effective mapping is to positional channels, i.e. coordinates in space. The units of raw data values rarely correspond exactly to the physical units we use to plot them on paper or on screen (e.g., pixels or points), so we need to rescale. As a simple example, let's make a scatterplot.

1. Open the template HTML code in your preferred code editor.

2. We are using an external JavaScript file to load D3. For security reasons, most browsers won't let us open the HTML file directly. To get our page running, we'll create a local server. Navigate to your folder in Terminal, and run the following command:
   ```
   python -m http.server
   ```

3. Open the link you get (probably `http://localhost:8000` or `http://[::]:8000/`) and navigate to the correct file. It should be a blank page, but as we write code we'll be able to access it here. When you want to quit your server, you can do so with the shortcut control-C in the terminal. If you run into issues, **this link** has a more detailed explanation of how to set up a local server.

4. Now copy and paste the following dataset into your file:
   ```
   var dataset = [
         [5, 20], [480, 90], [250, 50], [100, 33], [330, 95],
         [410, 12], [475, 44], [25, 67], [85, 21], [220, 88]
   ];
   ```

   To check if it worked, do the following:
   a) Save the file
   b) Refresh your page from the previous step
   c) Right click your page and click "inspect"
   d) Click the "console" tab

e) Type in `dataset` and press enter to see the contents of the variable you just created

5. Notice that if you were to directly draw the points, they'd exceed the canvas size of 400px by 400px. This is where a scale function becomes necessary. As an example of how it works, if you wanted to map the domain [100, 500] (i.e., the data range) to the range of [10, 350] (i.e., the pixel range) you could do:

```
var scale = d3.scaleLinear()
    .domain([100, 500])
    .range([10, 350]);
```

You can see how this works by putting it in your code, going back to the console tab (don't forget to save and refresh), and type in `scale(300)`, which should give you 180 (halfway between 10 and 350), because 300 is halfway between the domain 100 and 500. Try it out with some other numbers as well.

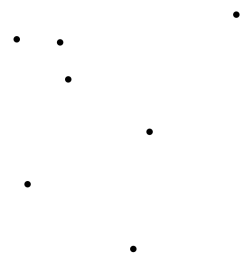
6. Create two variables xScale and yScale that map the dataset to the range [0, 400] (on the x-axis) and 0-400 (on the y-axis). Use `d3.max(...)` or `d3.extent(...)` to make this easier. You can look up these functions in the d3 API documentation. Use the console to see if they work as expected. You can also put `console.log(...)` in your code to print to the console.

If we think about what we are doing in terms of the **grammar of graphics**, we are creating functions that represent the mappings between data variables and visual channels: in this case, a mapping from `d[0]` → *x position* for all *d* in the dataset (this mapping is the function `xScale`), and a mapping from `d[1]` -> *y position* for all *d* in the dataset (this mapping is the function `yScale`).

7. Given the above scale functions (mappings from data onto visual channels), we will now instantiate those mappings in **marks** to be displayed on screen (in this case, `circle` elements in SVG). Paste this code in:

```
svg.selectAll("circle")       // Find all circle elements
   .data(dataset)             // Compare them against the dataset
   .enter().append("circle")  // Create new circle elements for
each data point
      .attr("cx", d => xScale(d[0]))  // Use the x and y mappings
(xScale/yScale) to
      .attr("cy", d => yScale(d[1]))  // determine the center of
the circle
      .attr("r", 5);                  // Set the radius to 5
```

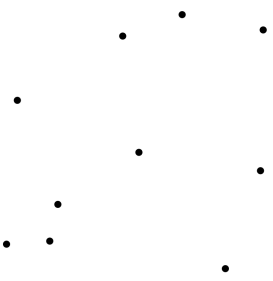
If you've done things correctly, you should see something like this (screenshot not to scale):



8. Change the code so that your origin, 0,0, which is currently in the upper left corner, is instead in the bottom left.

9. Your dataset has 10 values in it, but not all of them are showing up on the screen. Modify your xScale and yScale so they are entirely inside the drawing surface. If you're doing more work than changing a few numbers, you're probably doing too much work. Hint: CAREFULLY read all of the provided code and make sure you understand all of it.

   If you've done things correctly, you should see something like this (screenshot not to scale):

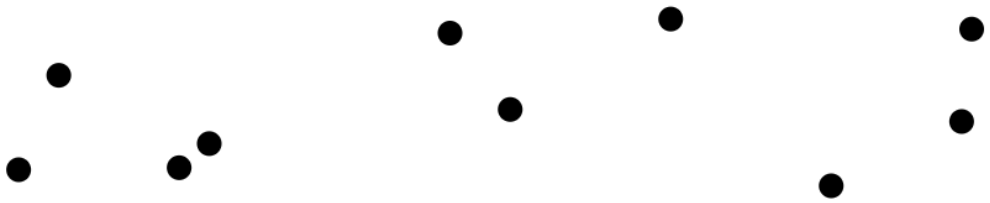


10. Save your file as part1.html and make sure your variables/functions are called dataset, xScale and yScale as specified above.

Part 2

1. Copy your file from part 1 and make a new one (call it part2.html).

2. Let's say we want to maintain the aspect ratio of the original dataset which is very long and not so tall. (screenshot not to scale)



What we want to do is use a scale to ensure that the maximum dimension (either width or height) fits inside 400x400. So if our picture looks like the one above, we'd want something like:

(where the width of this is 400).

If the dataset were pivoted (taller rather than wider), we'd want to see:

```
var dataset = [
      [20, 5], [90, 480], [50, 250], [33, 100], [95,330],
      [12, 410], [44, 475], [67, 25], [21, 85], [88, 220]
];
```

Modify your xScale and yScale to support this. You can assume a minimum value on both the x and y dimensions of the dataset to be 0, but you will need to make sure that you rescale both dimensions proportionally (i.e., if you had to shrink the x-direction 50% to make it fit, you should shrink the y dimension by that much).

3. Save your file as part2.html and make sure your variables/functions are called dataset, xScale and yScale as specified above and please document your code. Make sure you test for datasets that are bigger than the canvas space of 400x400 in each of the dimensions, both of the dimensions, or neither of the dimensions.

Part 3

1. Copy your file from part 2 and make a new one (call it part3.html).

2. Let's make a horizontal axis. Copy the following code into your file. Put it in the correct place in your code so that the axis is layered underneath the data points, not on top of them.

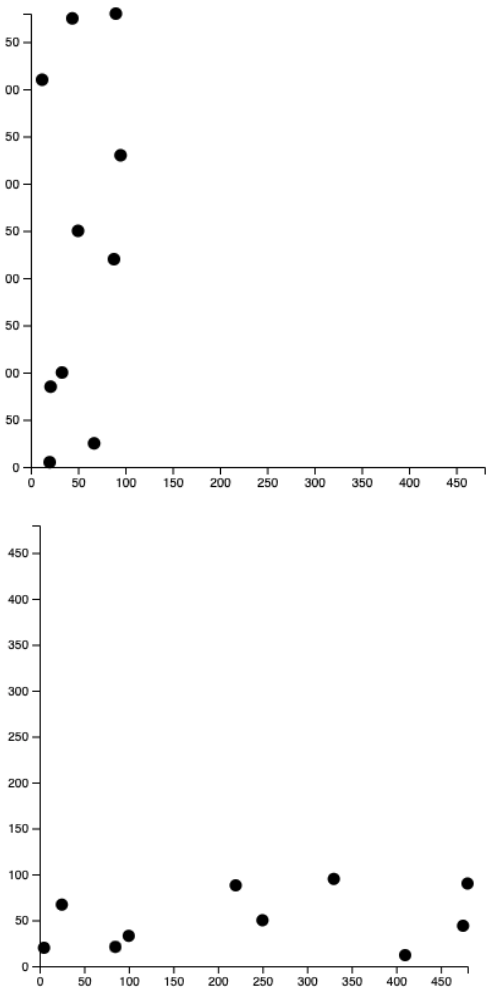
```
var xAxis = d3.axisBottom()
   .scale(xScale);

svg.append("g")
   .call(xAxis);
```

3. Because the origin is in the top-left corner, your axis is probably showing up on top of the page. Modify your code to move it to the correct place. Hint: CAREFULLY read all of the provided code and make sure you understand all of it.

4. Now add a vertical y-axis.

   If you've done things correctly, you should see something like one of these, depending on which version of `dataset` you're using.



5. Save your file as part3.html

Part 4

1. Copy your file from part 3 to a new file called part4.html

2. Copy this code to the bottom of your file.

```
// reset everything to color=black, radius=3
circles = svg.selectAll("circle")
   .data(dataset)
   .attr("fill","black")
   .attr("r",3);

// transition to magenta with a radius of 6 over 1 seconds
transition1 = circles.transition()
   .duration(1000)
   .attr("fill","magenta")
```

```
    .attr("r", 6);

// transition to black with a radius of 3 after a delay of 1
second
// and make the transition take 1 second
transition2 = transition1.transition()
    .delay(1000)
    .duration(1000)
    .attr("fill", "black")
    .attr("r", 3);
```

What this will do: reset everything to black with a radius of 3, smoothly transform the color to magenta with a radius of 6 and then to a black circle with a radius of 3. There are other ways to chain transitions, but this happens to be a simple one (that requires adding delays so the transitions run one after the other).

3. Modify the code so that in addition to doing the above, the points move to a random position. The new coordinates for a given point should be somewhere between 0 and the max X in the dataset, and 0 and the max Y in the dataset. You can make the coordinate change happen either in the first step (when it is going to magenta) or the second step (when it is going to black). Hint: look up Math.random() or look up the random number generating functions in D3, and look to the earlier code for guidance on how to manipulate individual points.

4. Make the last step happen over 3 seconds.

5. Save your file as part4.html and submit a zip file with all your files to Canvas.

D3 Resources

D3 API Reference: **https://github.com/d3/d3/blob/main/API.md**

D3 in Depth: **https://www.d3indepth.com/**

Observable D3 Tutorial: **https://observablehq.com/@d3/learn-d3**