Faculty of Science, Engineering and Technology

COS30045 Data Visualisation

Task 3.1 D3 Scaling your charts

ILO	Create web-based interactive visualisations using real-world data sets.
Aim:	Use D3 to make your charts more flexible by scaling the data from the input domain to fit your SVG canvas.
Resources:	*Textbook:* Murray Ch 7 Murray on ProQuest Murray on Safari (Make sure you use v2 of Murray as per links above) *Web Resources:* *JeromeCukier Tutorial on Scales and Colour*
To be marked as Complete your submission must:	Submit working code that meets the requirements specified in document below. Demonstrate appropriate use of HTML, CSS and D3. Properly formatted code Well commented code with references to code sourced from web, stack overflow etc. where appropriate. Demonstrate and explain code to tutor in class.
Submission	Submit to Doubtfire • screenshot of final webpage demonstrating the scaleability of your code • code Bring code to class to demonstrate to tutor

Note: The functions handling scale have changed between D3 v3 and D3 v4. This is something to be aware of if you are doing your on research into this topic. Make sure you use Murray Ed 2. Code examples from Ed 1 will not work.

Overview

In this tutorial we will start with your code from Task 2.3 (the scatter plot). At the end of this Task you should end up with a scatter plot drawn using D3 generated SVGs that looks something like this, no matter whether the min and the max are 10 points apart or 10,000 points apart. To do this we will use scales. "Scales are functions that map from an input domain to an output range." (Bostock, cited in Murray, 2016). It would be an unusual case if the range of our data set (aka the input domain) matched exactly with the number of pixels we had to display our chart (aka the output range). A D3 scale function maps the data set domain with the range we have available for our visualisation. Not only is it useful when we first create the visualisation, but using Scale functions makes our visualisation flexible enough to handle new data with wildly different min and max values. Figure 1 shows how the data point at 50 is mapped from a range of 20-80 to a range of 0-120.

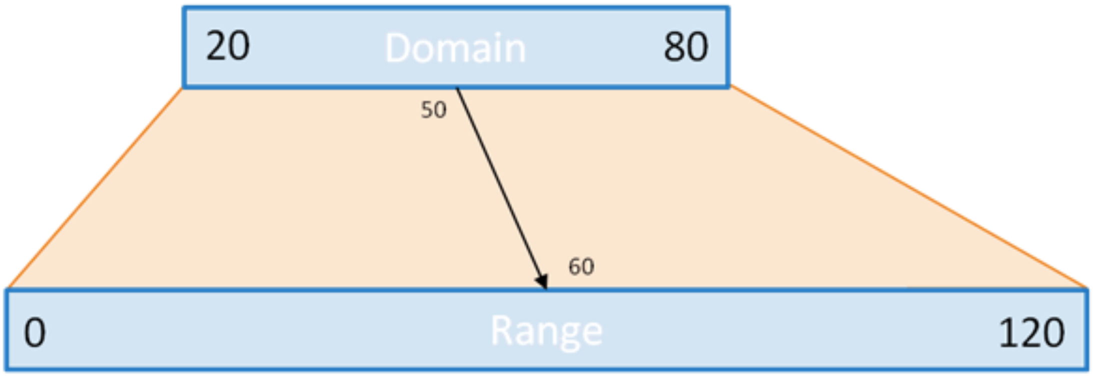
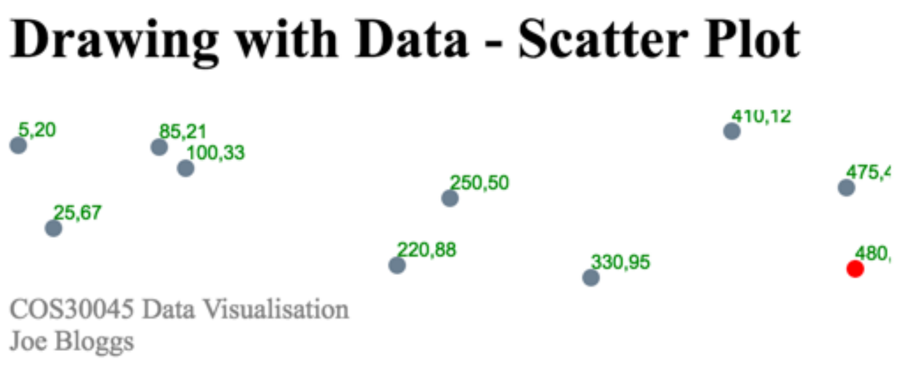


Figure 1: Domain and Range mapping From: JCukier Scales tutorial

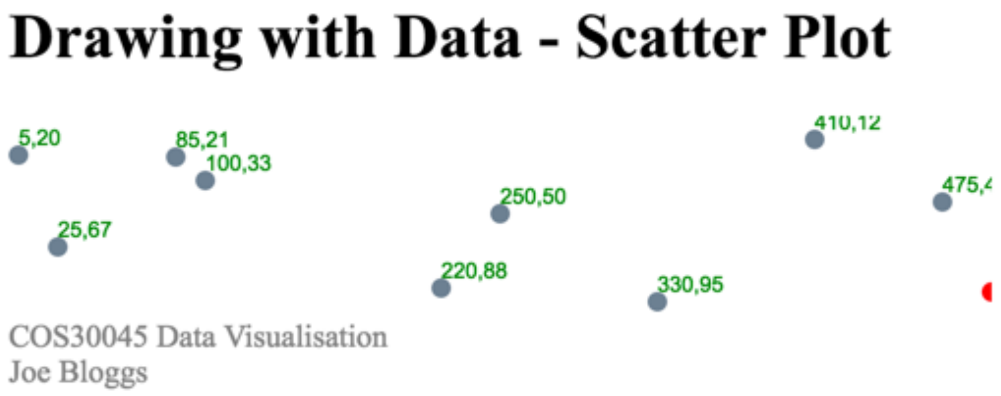
Note: Remember to put your D3 JavaScript code into a separate .js file and call it from the header.

Step 1: Start with the code from Task 2.3

We will start with the code from the previous scatter plot task, the one that should look a bit like this…



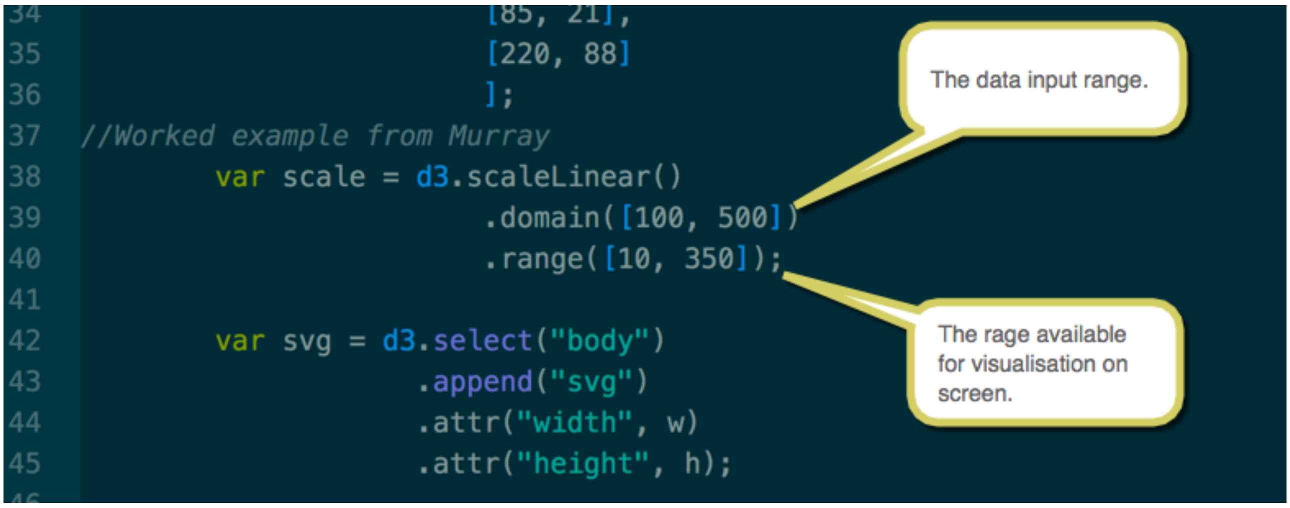
The width of this SVG is 500 pixels and the pixels and datapoint are mapped 1-1. If the [480, 90] data point is changed to [500, 90] this happens:



What we would prefer to happen is that the scale change to fit the data. We can make this happen by using the D3 `scaleLinear()` function.

Step 2 Create a Scale

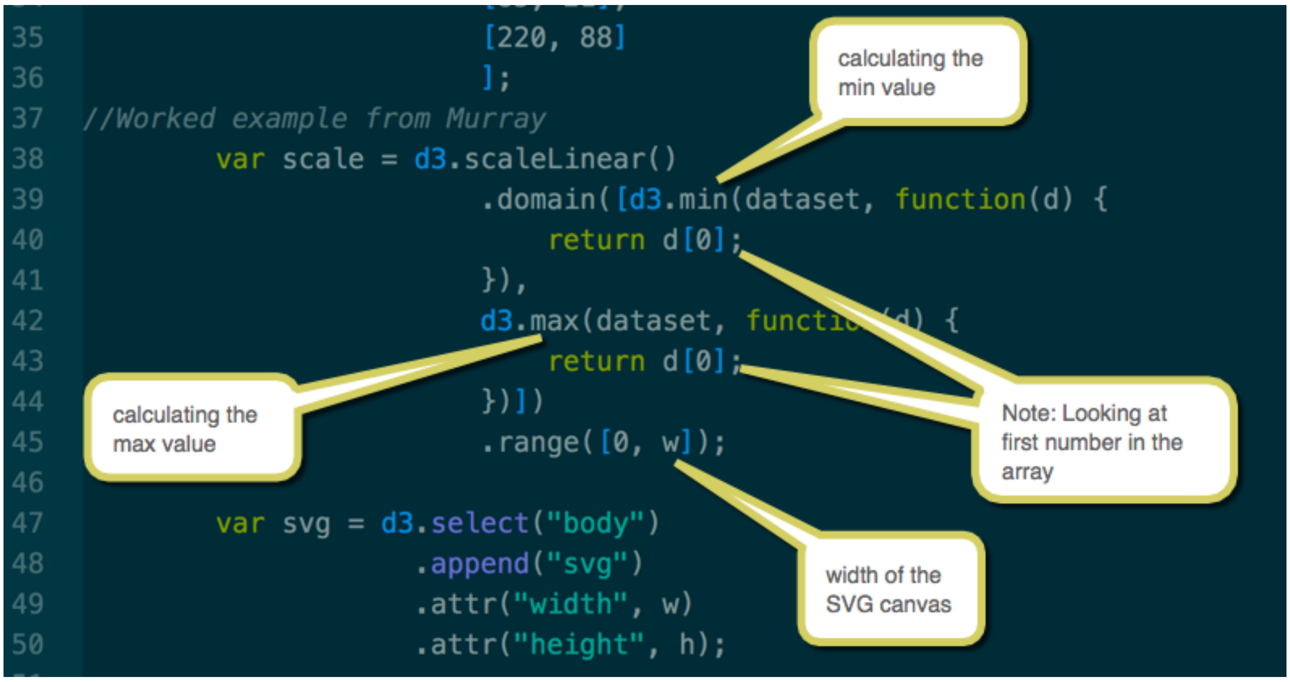
The first step in making our visualisation scalable is creating a scale. The `scaleLinear()` function allows you to specify the input domain and the output range as below:



```
                        [85, 21],
                        [220, 88]
                        ];
//Worked example from Murray
        var scale = d3.scaleLinear()
                        .domain([100, 500])
                        .range([10, 350]);

        var svg = d3.select("body")
                    .append("svg")
                    .attr("width", w)
                    .attr("height", h);
```

In this case we have hard coded the values. However, to make the code more flexible you can make use of the D3 min and max functions to find the actual range of the domain and the actual available space on the SVG canvas.



```
                        [220, 88]
                        ];
//Worked example from Murray
        var scale = d3.scaleLinear()
                        .domain([d3.min(dataset, function(d) {
                            return d[0];
                        }),
                        d3.max(dataset, function(d) {
                            return d[0];
                        })])
                        .range([0, w]);

        var svg = d3.select("body")
                    .append("svg")
                    .attr("width", w)
                    .attr("height", h);
```

calculating the min value
calculating the max value
Note: Looking at first number in the array
width of the SVG canvas

If you look carefully at the code above you will see that it’s calculating the min and max for the x value (i.e., the first data point in the array). However, we will also want to scale for the y value. Rename your first scale xScale and add in a second scale for the y value (i.e., yScale).

Step 3 Using the scales

So the scales are in place, but as yet they are not being used to generate the visualisation. Remember that D3 gets the x value for the circle positions from this bit of code:

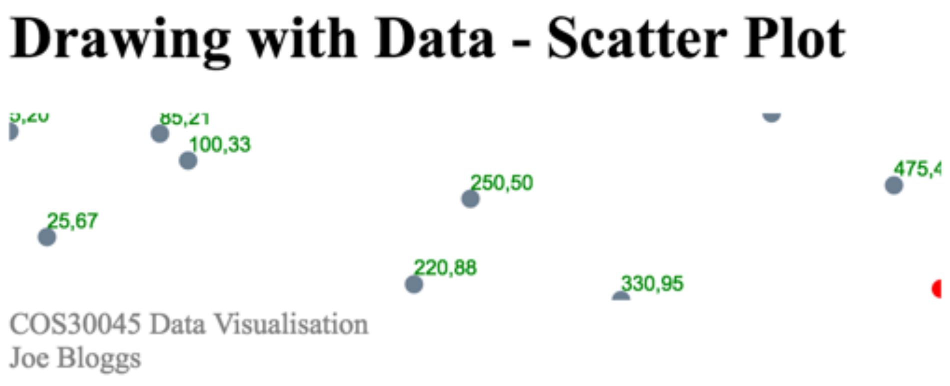
```
            .attr("cx", function(d, i) {
                return d[0];
            })
```

What we need to do is rather than read the actual data point, get the scaled data point:

```
            .attr("cx", function(d, i) {
                return xScale(d[0]);
            })
```

Do the same for the y value as well. We should also update the position of the text labels so they don't get left behind with changes to the data set!

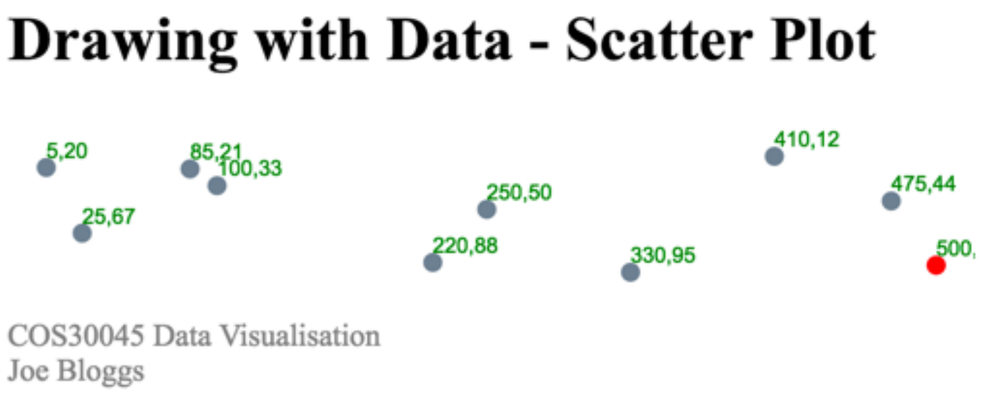
You should end up with something like this:



Unfortunately, some of our circles are getting cut off (remember the cx and cy specifies the centre of the circle). We can give visualisation a bit of space by introducing some padding. Add a padding variable to the w and h variables at the top of your script code. We can then use the padding variable to shorten the available screen space:

```
//Worked example from Murray
        var xScale = d3.scaleLinear()
                        .domain([d3.min(dataset, function(d) {
                            return d[0];
                        }),
                        d3.max(dataset, function(d) {
                            return d[0];
                        })])
                        .range([padding, w - padding]);
```

Do the same for the yScale and you should end up with something like this:



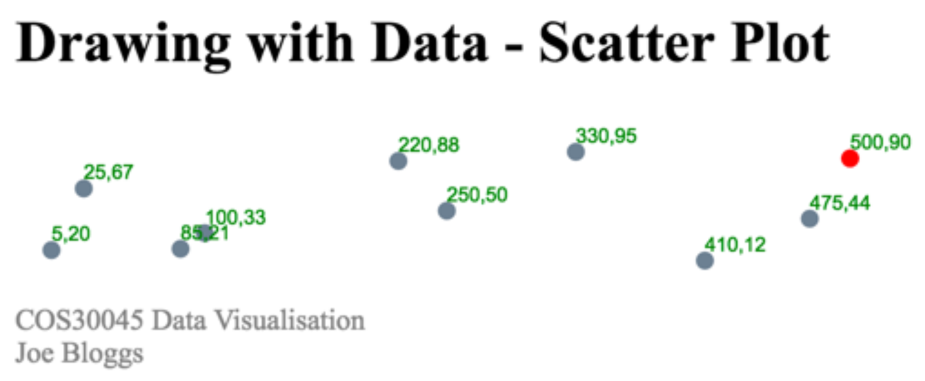
Drawing with Data - Scatter Plot

COS30045 Data Visualisation
Joe Bloggs

At the moment the high y values are at the bottom of the chart (remember the 0,0 point of a SVG is at the top left hand corner). You can reverse this changing the input range of the yScale.

```
.range([h - padding, padding]);
```

There is also still a few data babes that are getting cut off at the right hand end of the visualisation. Add a little bit more padding so they don't! You should end up with something like this.



Drawing with Data - Scatter Plot

COS30045 Data Visualisation
Joe Bloggs

Step 4 Demonstrate the flexibility of your code

Try adding some outliers to your data set and see what happens. Take a screen shot for your submission.

Try changing the size of your SVG and see what happens. Take a screen shot for your submission.