Tech Nomads

d3.js group coding session

Tutorial

- Taken from Scott Murray tutorial https://alignedleft.com/tutorials/d3/about

- Prerequisites for this course: use a text editor of your choice, I will be using visual studio code

Fundamentals I

- The following concepts are required when working with D3:

  - HTML: Hypertext Markup Language

  - DOM: Document Object Model refers to the hierarchical structure of HTML

  - CSS: Cascading Style Sheets used to style the visual presentation of HTML pages. Consists of selectors and rules (properties that form the styles)

Fundamentals II

- The following concepts are required when working with D3:
    - Javascript: dynamic scripting language that can instruct the browser to make changes to a page after it has loaded
    - Developer Tools: Be familiar with your browser's developer tools, the web inspector shows the current state of the DOM and the Javascript console can be used for debugging
    - SVG: Scalable Vector Graphics, a text-based image format and can be displayed within a HTML document

HTML

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Page Title</h1>
        <p>This is a really interesting
paragraph.</p>
    </body>
</html>
```

CSS

- Can be added to the document's head or an external .css file

```
body {
    background-color: white;
    color: black;
}

p {
    font-size: 12px;
    line-height: 14px;
    color: black;
}
```

Javascript

- Scripts can be included in HTML or stored in a separate file and referenced in the HTML

```
<body>
    <script type="text/javascript">
        alert("Hello, world!");
    </script>
</body>
```

SVG

```
<svg width="50" height="50">
    <circle cx="25" cy="25" r="22"
     fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```

Setup - downloading d3

1. Create a folder for your project
2. Create a subfolder called d3

3. Download the latest version of d3.v5.js into the subfolder (minified version is available, but best to use regular version when working on a project for friendlier debugging)

Setup - referencing d3

1. Create a simple HTML page within your project folder named `index.html`

2. Folder structure:
   project-folder/
   `d3/`
   `d3.v3.js`
   `d3.v3.min.js (optional)`
   `index.html`

3. Add the following to your HTML file - indexes the `head` and provides room for your Javascript code (see Github)

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Tech Nomads D3</title>
        <script type="text/javascript" src=“d3/d3.v5.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            // Your beautiful D3 code will go here
        </script>
    </body>
</html>
```

Adding Elements

- Getting started, we can use D3 to create a new DOM element. Typically this will be an SVG object for rendering a data visualisation, but to get started we will create a paragraph element

- Starting with the HTML template we have set up, replace the comment between the script tags with:

```
d3.select("body").append("p").text("New
paragraph!");
```

- Save and refresh the page to view, inspect the DOM and you should see the paragraph element and text!

Adding elements - summary

- What happened here?
    - We invoked D3's `select` method to select a single element from the dom, in this case the `body`
    - Created a new `p` element and appended that to the end of our selection
    - Set the text content of the new, empty paragraph to "New paragraph!"
- The dots we have used are part of D3's chain syntax

Chaining methods

- Similar to jQuery, D3 uses a technique called chain syntax

- By chaining methods, we can perform several actions in a single line of code

- Javascript, like HTML does not care about whitespace or line breaks so for readability we can put each method on a new line

- The order of your chain matters! The output type of one method must match the input type for the next method in the chain!

Alternatively…

- We can instead break the chain which can be useful when chaining many functions by declaring each function as a

```
var
```

```
var body = d3.select("body");
var p = body.append("p");
p.text("New paragraph!");
```

Binding Data

- Data visualisation is a process of mapping data to visuals
- With D3, we bind our data input values to elements in the DOM so that later you can reference those values to apply mapping rules
- We use D3's `selection.data()` method to bind data to DOM elements, we therefore require:
  - The data
  - A selection of DOM elements

Data

- D3 can accept many types of data, it can handle arrays of numbers, strings or objects (containing other arrays or key/value pairs), JSON, GeoJSON and can input csv files

- Let's start with an array of numbers:

```
var dataset = [ 5, 10, 15, 20, 25 ];
```

Elements

- We need to decide what we want to select - which elements do we want the data to be associated with?

- If we use the following:

```
d3.select("body").selectAll("p")
```

- The elements we are trying to select don’t exist yet.

- D3 has a function `enter()` which creates a new placeholder elements then hands off a reference to this new placeholder to the next step in the chain

Bind data to elements

- Data can been read, parsed and bound to new p elements we create in the DOM using:

```
d3.select("body").selectAll("p")
  .data(dataset)
  .enter()
  .append("p")
  .text("New paragraph!");
```

- We can inspect the data using the console:

```
console.log(d3.selectAll("p"))
```

Bind data to elements

- Here the text has repeated however, we can use our data to populate the contents of each paragraph with:

```
    .text(function(d) { return d; });
```

- Asasda

Adding style

- D3's other methods like `attr()` and `style()` allow us to set HTML attributes and CSS properties on selections

- For example we can change the colour with:

```
.style("color", "red");
```

- Or use a custom function to change the colour if we value exceeds a certain threshold

```
.style("color", function(d) {
    if (d > 15) {   //Threshold of 15
        return "red";
    } else {
        return “black";
    }
});
```

Drawing divs I

- Using our data from before, let's start to make a bar chart

- Bar charts use rectangles, the simplest way to draw a rectangle in HTML is using a `<div>`

```
<div style="display: inline-block;
            width: 20px;
            height: 75px;
            background-color: teal;"></div>
```

- The width and height of the element is set by CSS styles, we can put the shared styles into a class called bar

```
div.bar {
    display: inline-block;
    width: 20px;
    height: 75px;   /* We'll override this later */
    background-color: teal;
}
```

Drawing divs II

- Need to assign each div to a bar class, using HTML we would use:

```
<div class="bar"></div>
```

- Using D3, add a class to an element using the `selection.attr()` method

- Setting an attribute differs from applying a style directly to an element, useful to use classes for properties that are shared by multiple elements

- D3 method `classed()` can be used to quickly apply/remove classes from elements

```
.classed("bar", true)
```

Drawing divs III

- To create our first bar, with the CSS we have already set up, we can use the following:

```
d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar");
```

- Viewing the page looks like one bar but inspecting the DOM you can see that 5 bars have been created

- In a bar chart, each bar must be a function of the corresponding data value, we can use the style() method to set the height of each bar to the value of the data point

```
    .style("height", function(d) {
        return d + "px";
    });
```

- To increase the size of the bars, you can apply a factor of 5 to the hight and also appy a margin in the CSS to separate the bars with:

```
margin-right: 2px;
```

Using data

- So far we have used a simple array as our dataset but we can easily add new data points, increase our dataset for example:

```
var dataset = [ 25, 7, 5, 26, 11, 8, 25, 14, 23, 19,
                14, 11, 22, 29, 11, 13, 12, 17, 18, 10,
                24, 18, 25, 9, 3 ];
```

- D3 automatically expands our chart since the `.data()` function is looping through each data point and d always refers to the current data at that point in the loop

- We can use JavaScript to generate some random data points:

```
var dataset = [];                        //Initialize empty array
for (var i = 0; i < 25; i++) {           //Loop 25 times
    var newNumber = Math.random() * 30;  //New random number (0-30)
    dataset.push(newNumber);             //Add new number to array
}
```

- We can view the data values in the console with console.log(dataset)

Intro to SVGs

- D3 is most useful when used to generate and manipulate visuals as SVGs, an XML-based image format - this means that elements are self-closing!

- With SVG it is useful to specify the width and height values, eg.

```
<svg width="500" height="50">
</svg>
```

- Visual elements of SVG include; `rect`, `circle`, `ellipse`, `line`, `text` and `path`

- The coordinate system origin is in the top left with `x` increasing to the right and `y` increases down the page

- Note when any visual elements runs up against the edge of an SVG, it will be clipped

- There are no layers in SVG and no real concept of depth, it does not support CSS's z-index property so shapes can only be arranged in 2D x-y plane - therefore, the order of elements determine their depth order

- Check out this page for more details on SVGs!

Drawing SVGs I

- All properties of SVG elements are specified as attributes - included as property/value pairs within each element tags

```
<element property="value"/>
```

- To create the SVG elements we can use:

```
d3.select("body").append("svg");
```

OR we can create an SVG object

```
var svg = d3.select("body").append("svg");
```

- We could also chain several methods together to create the SVG by first declaring the width and height values at the top of your code:

```
// Width and height
var w = 500;
var h = 50;
```

- Then creating the SVG:

```
var svg = d3.select("body")
            .append("svg")
            .attr("width", w)   // <-- Here
            .attr("height", h); // <-- and here!
```

Drawing SVGs II

- To then create shapes using SVG and our simple dataset:

```
var dataset = [ 5, 10, 15, 20, 25 ];
```

- To make it easier to reference all of the circles later, we can create a new variables to store references to them all:

```
var circles = svg.selectAll("circle")
                 .data(dataset)
                 .enter()
                 .append("circle");
```

- The circles then need positions and sizes to be set using:

```
circles.attr("cx", function(d, i) {
            return (i * 50) + 25;
        })
       .attr("cy", h/2)
       .attr("r", function(d) {
            return d;
       });
```

- If we want to add some style, we can add:

```
.attr("fill", "yellow")
.attr("stroke", "orange")
.attr("stroke-width", function(d) {
    return d/2;
});
```

Data types

- D3 is extremely flexible with input data..

- **Variables** - JavaScript is a loosely typed language, automatically types a variable based on the input

```
var number = 5;
var value = 12.3467;
var active = true;
var text = "Crystal clear";
```

- **Arrays** - store comma separated variables as we have been using!

```
var numbers = [ 5, 10, 15, 20, 25 ];
```

- **Objects** - with most complex datasets we can use objects with curly brackets to indicate an object

```
var fruit = {
    kind: "grape",
    color: "red",
    quantity: 12,
    tasty: true
};
```

- **JSON** - similar to what we've already seen, uses double quotation marks

```
var jsonFruit = {
    "kind": "grape",
    "color": "red",
    "quantity": 12,
    "tasty": true
};
```

Bar chart I

- Tying together what we've covered so far we can generate a simple bar chart with D3
- Review the bar chart we made earlier with the div elements, we can adapt the code to use SVGs

    1. Set the width and height of the SVG object

    2. Create an SVG object

    3. Create `var rect` with the following:

```
svg.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("x", 0)
   .attr("y", 0)
   .attr("width", 20)
   .attr("height", 100);
```

    4. Check the DOM to see the 20 bars, all with the same x, y, width and height values

    5. Assign attribute x:

```
.attr("x", function(d, i) {
    return i * (w / dataset.length);
})
```

Bar chart II

6. The bars are evenly spaced but the width is constant. Add a new variable where we have declared the width and height and set the width of the rects to be:

```
var barPadding = 1;
.attr("width", w / dataset.length - barPadding)
```

7. Change the height of each bar to represent our data:

```
.attr("height", function(d) {
    return d;
});
```

8. We want the bars to be a bit taller, scale by 4

9. To start the bars at the bottom rather than the top of the page, we can redefine y to be:

```
.attr("y", function(d) {
    return h - d;  //Height minus data value
})
```

10. Finally, scale this value by 4 and check out your results!

Bar chart style

- To change the colour of the bars we can use the fill method - have a test and change your bar chart colours!

- Another option is to write a custom function:

```
.attr("fill", function(d) {
    return "rgb(0, 0, " + (d * 10) + ")";
});
```

- To give further insight to the data, labels are useful! We can use SVG text elements:

```
svg.selectAll("text")
   .data(dataset)
   .enter()
   .append("text")
```

- Extending the code to include a data value and x, y values to position the text

```
.text(function(d) {
      return d;
 })
 .attr("x", function(d, i) {
      return i * (w / dataset.length);
 })
 .attr("y", function(d) {
      return h - (d * 4);
 });
```

- Try adding an offset to the x and y positions so that the text lies inside the bars!

- Finally to make this more legible and stylistic we can change the font attributes:

```
.attr("font-family", "sans-serif")
.attr("font-size", "11px")
.attr("fill", “white");
.attr("text-anchor", "middle")
```

Scatterplot I

- When you have to sets of values to plot, a scatterplot is a common type of visualisation to use

- For our dataset we will use an array of arrays

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

- Using the same steps as when we made the bar chart

1. Create an SVG element

2. Create an SVG circle with the following attributes:

```
.attr("cx", function(d) {
     return d[0];
})
.attr("cy", function(d) {
     return d[1];
})
.attr("r", 5);
```

3. Check out your result!

Scatterplot II

- The x, y position of the circles is taken from the data
- We can also change the size of the circles by changing the radius, for example

```
.attr("r", function(d) {
    return Math.sqrt(h - d[1]);
});
```

- To then add axes to the

Scales

- To convert between your data values and pixels, scales are used to map the data values to new values

- With a linear scale, this is simply normalisation - mapping a numeric value to a new value between 0 and 1 and we let D3 handle the math of normalisation

- The input value is normalised according to the domain, and the normalised value is scaled to the output range

- D3 scale generators are accessed using:

```
var scale = d3.scaleLinear();
```

Scaling the scatterplot

- To analyse our dataset, we can use the linear scale function to redefine the xy positions and size of the data points:

```
var xScale = d3.scaleLinear()
                     .domain([0, d3.max(dataset, function(d) { return d[0]; })])
                     .range([0, w]);
var yScale = d3.scaleLinear()
                     .domain([0, d3.max(dataset, function(d) { return d[1]; })])
                     .range([0, h])
var rScale = d3.scaleLinear()
                     .domain([0, d3.max(dataset, function(d) { return d[1]; })])
                     .range([2, 5]);
```

- Change the x, y and r attributes to call these functions

- You may notice that the edges of our circles are getting cut off, try introducing a padding variable of 20 and change the range of x, y to incorporate the padding (remembering to also switch the y height to start from the bottom of the page!)

Adding axes I

- Finally, we can add axes to the scatterplot using SVG elements! We can add the axis with a scale:

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

- Apply the same to create a y-axis

- To generate the x axis and insert into the SVG, we need to call the `xAxis` function

```
svg.append("g")
    .attr("class", "axis")  //Assign "axis" class
    .call(xAxis);
```

- The `call()` function takes a selection as input, in this case we have appended a new `g` group element, can can also assign an axis class to introduce CSS styles

```
.axis path,
.axis line {
    fill: none;
    stroke: black;
    shape-rendering: crispEdges;
}

.axis text {
    font-family: sans-serif;
    font-size: 11px;
}
```

Adding axes II

- The axis is positioned at the top of the page and to move it to the bottom we can `transform` the axis group, pushing it to the bottom (before calling xAxis)

```
.attr("transform", "translate(0," + (h - padding) + ")")
```

- The number of ticks on the axis can be roughly set with:

- The same method can be used for the y axis, be sure to use the left axis and think about how to transform this axis

- Try increasing the padding and check your plot!

Adding more data

- Finally, add some dynamic, random dataset and check how your scatterplot changes!

```
//Dynamic, random dataset
var dataset = [];
var numDataPoints = 50;
var xRange = Math.random() * 1000;
var yRange = Math.random() * 1000;
for (var i = 0; i < numDataPoints; i++) {
    var newNumber1 = Math.round(Math.random() * xRange);
    var newNumber2 = Math.round(Math.random() * yRange);
    dataset.push([newNumber1, newNumber2]);
}
```

Create your own

- This tutorial was adapted from Scott Murray's tutorials: https://alignedleft.com/tutorials/d3

- Project:

1. Use your own data, find data from kaggle (https://www.kaggle.com/datasets) and use `d3.csv()` to create your own visualisations

2. Choose a visualisation from https://github.com/d3/d3/wiki/Gallery and recreate/create your own

3. Share your visualisations with us on twitter @technomads