CASA0003: Digital Visualisation
Interactive Charts Practical

This practical introduces ways of creating online charts for data visualisations. There are many libraries and tools available to create online charts. These options can be summarised into three general groups-

Cloud Service Tools
Easy-to-use online commercial services for creating interactive charts based on templates (similar to what Mapbox provides for online maps). The leader is Tableau, and there are also more recent Data Journalism options such as Flourish and DataWrapper. These tools provide graphical interfaces for creating charts, can host your data and include many different chart types. Generally, these services cost money (though may have free trial options), and there can be limits in terms of the options for customising charts.

JavaScript Chart Libraries
There are many libraries available to create charts in JavaScript. Popular tools include Plot.ly, High Charts and Google Charts. We will use one of the simpler examples of these called Dimple.js in this practical. These libraries offer templates of common chart types that can be customised to your own data. There are generally free libraries, while there are also some commercial options too. Some more sophisticated libraries also have Python and R versions as well.

Web Data Visualisation Libraries
These are more fundamental libraries for manipulating data and developing charts and other types of data visualisation. D3 (Data Driven Documents) is the most popular of these. These tools are highly customisable, able to create many different chart types and even create new types of charts. The downside is that the coding requirements are more substantial. We will look at some D3 examples in this practical.

Importing CSV Data for Online Charts

The example HTML files and data for this practical are on Moodle. As with the online mapping examples in the previous practicals, the data for online charts is generally loaded from an external source. Separating the data from the design is good practice, and improves the reusability of your code. Data for online charts can be in the form of a CSV file or JSON (JavaScript Object Notation) file for archived data; or can involve requests to an API or web server for live data (more on this in Week 6).

For this practical we are going to use data from a CSV file (CityData_WUP2018.csv). This is a dataset of the largest urban agglomerations in the world according to the United Nations World Urbanization Prospects 2018. There are three versions of this CSV included- all 2000 cities, the top 200 and the top 20. To access this data, we need to load this CSV file using JavaScript.

Take a look at example 2, “CSVdata_example2.html”. This example imports the CSV data using JavaScript. The import is handled using a method from D3, d3.csv-

d3.csv("CityData_WUP2018_top20.csv", function(CityData) {…}

D3.csv handles the file request (HTTP Get), parses the CSV data to the variable CityData, and then runs a function when the CSV data is loaded. This is another example of an asynchronous JavaScript command that runs after data has loaded, just like the map.on('load'…) function that we used in the Mapbox examples. To find out more about the D3 methods for loading data (such as d3.xml, d3.json and d3.tsv), see-
https://www.tutorialsteacher.com/d3js/loading-data-from-file-in-d3js

D3 creates an array from the CSV data, where each item in the array is a JavaScript object representing one row of the CSV table. JavaScript object data uses the JavaScript Object Notation or JSON format, where each item is composed of a name and value pairs- {"name": "value"}. The code below shows how to extract the data using the data item name, or using a for loop.

CSVdata_example2.html

```
<!DOCTYPE html>
<html>
<head>
<title>D3 Load CSV Example</title>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

<!--Load the chart libraries. Dimple is built on D3, and you need to also add D3-->
<script charset="utf-8" src="https://d3js.org/d3.v4.min.js"></script>


</head>

<body>

<div id="demo"></div>


<script>


//The following statement loads the data from the CSV file, and then runs the function after the CSV is loaded. CityData is returned as an array of JavaScript objects containing the csv data

    d3.csv("CityData_WUP2018_top20.csv", function(CityData) {

        console.log(CityData);
        console.log(CityData[1]);

        var myData = CityData[1];    // Take the data for the second city in the array, Delhi

        console.log(myData["CityName"]);
        console.log(myData.CityName);
        console.log(myData.pop2020);

      // You can also loop through JSON Objects with a for loop-
      for (x in myData) {
        document.getElementById("demo").innerHTML += "<p>" + x + ": " + myData[x] + "</p>";
      }

    });
```

```
</script>

</body>
</html>
```

However when you try to run the above code, you likely hit a security error and nothing happens in the browser. For example, if you run the local version of this example using the Chrome browser you will get the following error in the developer console-

The 'CORS policy' (Cross Origin Requests) blocks access to local files as a security precaution, and the visualisation will not work. This is because web security blocks access to local files to prevent malicious online code. To overcome the CORS error, you can try the following solutions-

Live Preview in Brackets, and Live Server in VS Code

You can get around the CORS error by running a local host. Brackets has this feature built in by pressing the 'Live Preview' lightning button on the top right-

This will run the html page in your default browser with a local host. The equivalent feature in VS Code requires adding an extension called Live Server-
https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer

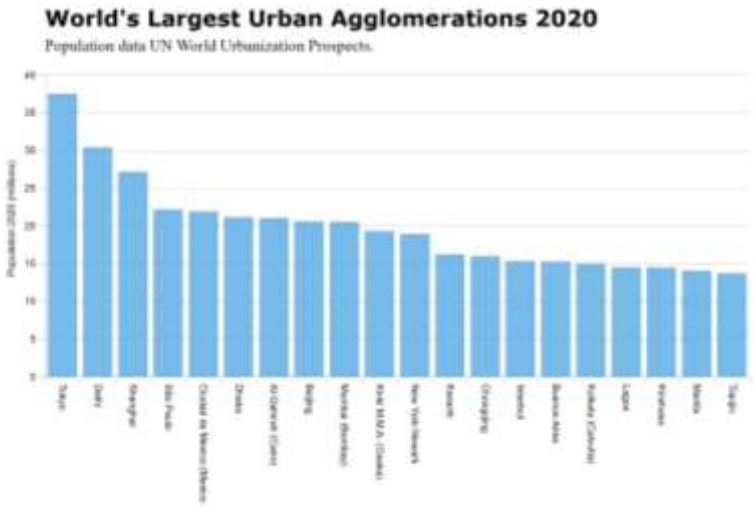
Uploading to a Web Server

To make your websites public, you need to upload your page to a web server. This also avoids any CORS errors, providing the HTML pages and the data are in the same directory (to access data from another server, you need to use an API- discussed next week). Github Pages is a good free option for static websites that we will be using later in this module.

Creating a Bar Chart in Dimple.js

Dimple.js is a JavaScript charting library built on D3. It offers a series of standard chart types with relatively minimal coding required: http://dimplejs.org/examples_index.html

Our first example is a Dimple bar chart. This example uses the 'CityData_WUP2018_top20.csv' to plot the population in the top 20 largest urban agglomerations according to the UN-



We need to add the Dimple.js library and the D3.js library in the header in this example. The CSV data is loaded using the d3.csv method described previously. The CSV data is loaded as the variable CityData, which is an array of JSON objects. Dimple, like D3, is based on a web standard for vector graphics called SVG (Scalable Vector Graphics). An SVG variable is created, with the width and height for the chart. Then the myChart variable is created using the svg variable and the CityData variable as arguments.

We then create the x and y axes, using the column names from the CSV file. The 'addseries' statement is used to define the chart type- 'dimple.plot.bar'. Finally, the chart is plotted using

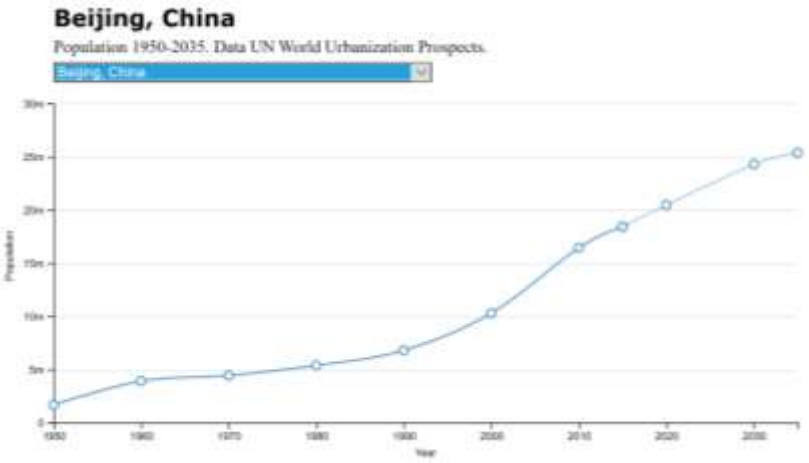
“myChart.draw(500);”. The argument here is a delay in milliseconds used for an animation effect. This can be used for transitions between different data series in more advanced examples.

Dimple_barchart_example3.html

```
<!DOCTYPE html>
<html>
<head>
<title>World City Populations Chart 2020</title>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

<!--Load the chart libraries. Dimple is built on D3, and you need to also add D3-->
<script charset="utf-8" src="https://d3js.org/d3.v4.min.js"></script>
<script charset="utf-8" src="https://cdnjs.cloudflare.com/ajax/libs/dimple/2.3.0/dimple.latest.min.js"></script>


<style>

   #chartContainer {width: 840px; margin: auto; margin-top: 50px; }
   #chartContainer p {margin-left: 60px; margin-bottom: 5px; margin-top: 0; padding: 0;}
   #charttitle {font: bold 22px  Verdana, sans-serif;}

</style>
</head>

<body>

<!--This is the div container for the chart-->

<div id="chartContainer">
   <p id="charttitle">World's Largest Urban Agglomerations 2020</p>
   <p id="chartsubhead">Population data UN World Urbanization Prospects.</p>
</div>

<script>

//The following statement loads the data from the CSV file, and then runs the function after the CSV is loaded. CityData is returned as an array containing the CSV data

    d3.csv("CityData_WUP2018_top20.csv", function(CityData) {

        var svg = dimple.newSvg("#chartContainer", 840, 440); // The chart is an svg variable assigned to the chartcontainer div. It's width and height are also assigned here

        var myChart = new dimple.chart(svg, CityData);  // Create the chart with CityData as the data input
        myChart.setBounds(60, 15, 700, 300);          // Set the chart bounds within the svg container, top-left and bottom-right coords measured from top left

        myChart.defaultColors = [
           new dimple.color("#54aae3")
        ];


        var x = myChart.addCategoryAxis("x", "CityName");  // Define the x axis. In this example it is a category axis

        var y = myChart.addMeasureAxis("y", "pop2020"); // Define the y axis
        y.title = "Population 2020 (millions)";

        var s = myChart.addSeries(null, dimple.plot.bar); // Plot a bar chart of the data
```

```
        myChart.draw(500); // Draw the chart. The number is the animation delay in miliseconds

        svg.selectAll("path.domain").style("stroke", "#CCC"); // These statements change the chart gridlines to a
lighter grey colour
        svg.selectAll("g.tick line").style("stroke", "#CCC");

    });

</script>

</body>
</html>
```

The second Dimple example, *Dimple_linechart2.html*, shows a line chart example using the same library. This time the line chart shows the change in population for a single city. Dimple requires time series data to be formatted as a separate row for each time point. So in this example, we create a new chartdata JavaScript Object Notation (JSON) array that is populated using the original CityData array. Each row is a different decade in the time-series. The final Dimple example, *Dimple_linechart2b.html*, adds the ability for users to change which city is plotted in the line chart example, using an HTML dropdown menu (select element) and an event listener.



An Introduction to D3

The Dimple.js library generally simplifies the process of creating standard charts online. On the downside, Dimple.js is limited to the more basic chart types. For more advanced charts and more control over customisation, it can be useful to use the underlying D3 library directly. D3 (Data Driven Documents) is a popular library for web visualisation created by the Programmer and Data Journalist Mike Bostock. It is based on the open web standards HTML, SVG and CSS, and offers powerful capabilities for binding data to web objects using the Document Object Model (DOM). There are a large number of both basic and advanced examples of different charts and visualisations created using D3 which can be viewed online- https://github.com/d3/d3/wiki/Gallery

More recently, D3 has also developed an editable notebooks version for easy sharing of templates called Observable.

D3 is a powerful tool, though has quite a substantial learning curve. We will only cover the basics in this practical. If you want to develop your skills further, some useful tutorials can be found in the following locations-

D3 API Reference- https://github.com/d3/d3/blob/master/API.md

D3 Tutorials links- https://github.com/d3/d3/wiki/Tutorials

Some Basic SVG Shapes- Example 5

The underlying standard for rendering graphics in D3 is SVG. SVG uses the Document Object Model (DOM) just like HTML, and has a similar structure of tag elements. Basic shape elements include <rect> for rectangles and <circle> for circles. The example below creates some basic rectangles that will underlie a bar chart. Note the coordinates are defined from the top-left corner of the SVG element.

D3_example5.html

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>D3 Example 5- Core SVG Elements</title>
        <!-- You will load the D3 library here. -->
</head>
<body>
        <!-- Page elements and content go here. -->
        <svg width="320" height="360">
                <rect x="30" width="40" height="80" fill="steelblue"></rect>
                <rect x="75" width="40" height="40" fill="steelblue"></rect>
                <rect x="120" width="40" height="140" fill="steelblue"></rect>
                <rect x="165" width="40" height="80" fill="steelblue"></rect>
        </svg>
        <script>
        <!-- Our D3 code will go here. -->
        </script>
```

```
</body>
</html>
```

Manipulating SVG Elements Using D3- Example 6

The first example hardcodes the SVG elements. In this next example, the height of the rectangles is defined in the D3 code. We create an array called popData with the values for our bar chart. The D3 script begins with a d3.selectAll("rect") method. This selects all the rectangle elements and iterates through each element. The data() method is central to how d3 works, and here the .data(popData) line binds the data array to the rect elements. Next the ".attr("height", function(d){ return d;});" line changes the height value to the corresponding value in the popData array.

D3_example6.html

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>D3 Example 6- Apply height of rectangles from array data</title>
        <!-- Load the D3 library here -->
        <script src="https://d3js.org/d3.v4.js" charset="utf-8"></script>
</head>
<body>
        <!-- Page elements and content go here. -->
        <svg width="320" height="360">
                <rect x="30" width="40" fill="steelblue"></rect>
                <rect x="75" width="40" fill="steelblue"></rect>
                <rect x="120" width="40" fill="steelblue"></rect>
                <rect x="165" width="40" fill="steelblue"></rect>
        </svg>
        <script>
                <!-- D3 code here. -->
                var popData = [ 80, 40, 140, 120 ]; // Array with graph data

                d3.selectAll( "rect" ) // select all rectangle elements
                        .data( popData )
                        .attr( "height", function(d){ // assign the data to the attribute height
                                return d;
                        });

        </script>

</body>
</html>
```

It is more common to create the entire SVG html code in D3 and have no SVG elements hardcoded. This can be seen in example 3. In this example we select the body element and append the SVG element in the D3 code. The next stage is a bit counter-intuitive, as we still select all the rectangle elements (if these do not exist, then D3 creates them) even though the rectangle elements have not been created in the HTML. Example 3 also includes code to set the coordinates of the bars as a function based on the number of elements in the array, and to align the bars at the bottom of the chart, rather than at the top-

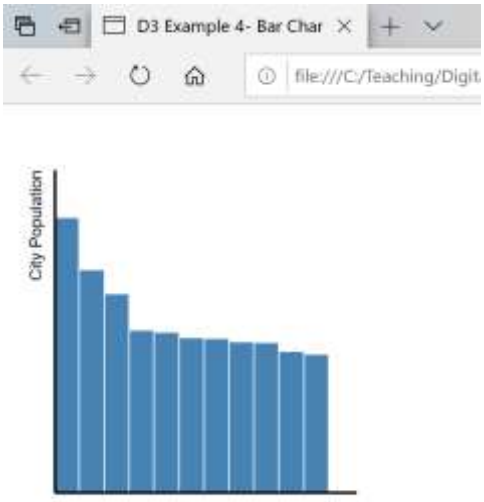
D3_example7.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
	<title>D3 Example 7- Basic Bar Chart</title>
	<!-- Load the D3 library here -->
	<script src="https://d3js.org/d3.v4.js" charset="utf-8"></script>
</head>
<body>

<script>
	// Our D3 code will go here.
	var popData = [ 80, 180, 60, 120 ]; // Data used for the bar chart

	// Width and height of SVG
	var w = 300;
	var h = 350;

	//Create SVG element
	var svg = d3.select("body")
			.append("svg")
			.attr("width", w)
			.attr("height", h);

	// Select and generate rectangle elements
	svg.selectAll( "rect" )
			.data( popData )
			.enter()
			.append("rect")
			.attr( "x", function(d,i){
				return i*50 + 30; // Set x coordinate of rectangle to index of data value (i)*25.
				// Add 30 to account for our left margin.
			})
			.attr( "y", function(d){
				return h - d; // Set y coordinate for each bar to height minus the data value
			})
			.attr( "width", 40 )
			.attr( "height", function(d){
				return d; // Set height of rectangle to data value
			})
			.attr( "fill", "steelblue");
</script>



</body>
</html>
```

Creating a Barchart from the CSV File in D3- Example 8

In this example, we fill the popData array using the data from a CSV file (CityData_WUP2018_top20.csv- populations from the top 20 cities). This is similar to the first Dimple.js example where we used the d3.csv() method, which returns the CSV data as a JSON array. We also define the position of the bars based on the length of the data array. Finally, we add a basic x and y axis, and y axis title.



D3_example8.html

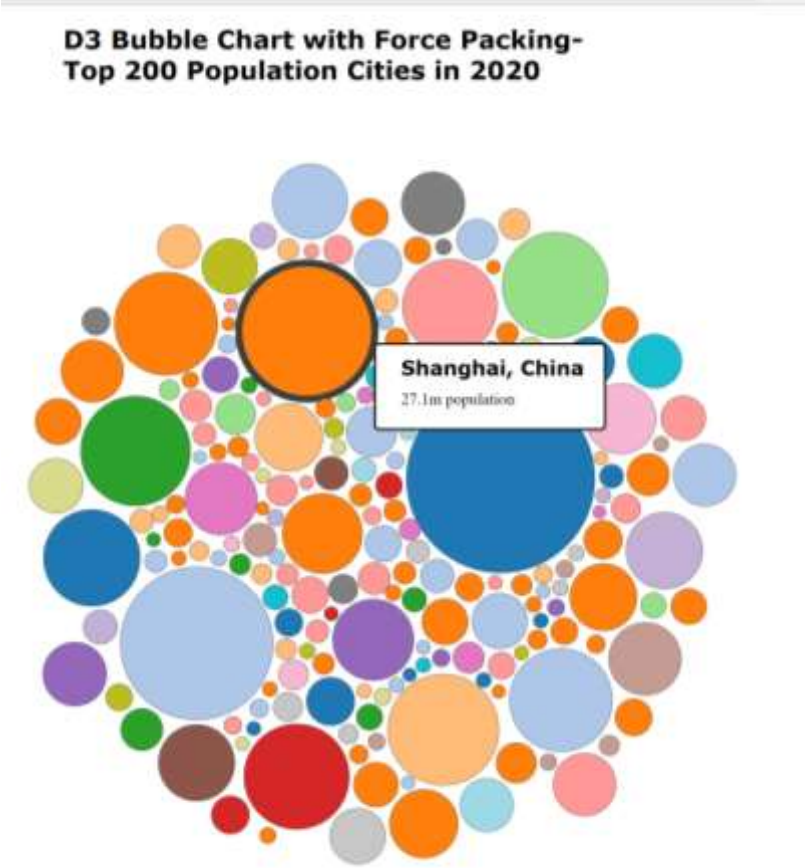
```
<!DOCTYPE html>
<html lang="en">
<head>
	<title>D3 Example 8- Bar Chart Data from CSV</title>
	<script src="http://d3js.org/d3.v4.min.js" charset="utf-8"></script>
	<style>
		text {
			font-family: "Open Sans", sans-serif;
			font-size: 12px;
		}
	</style>
</head>
<body>
	<!-- Location for page elements. -->
<script>
	// Our D3 code will go here.
	var popData = [];

	d3.csv("CityData_WUP2018_top20.csv", function(d) {
		return {
			CityName : d.CityName,
			pop2020 : +d.pop2020  // the + sign forces the values to be a number
		};
	}, function(error, rows) {
		popData = rows;
		console.log(popData);
		createVisualization();
	});

	function createVisualization(){
		// Width and height of SVG
		var w = 300;
		var h = 350;

		// Get length of Array and set length so we can an input dataset of variable length
		var arrayLength = popData.length; // length of dataset
		var maxValue = d3.max(popData, function(d) { return +d.pop2020;} ); // get maximum value of our
dataset
    var x_axisLength = 200; // length of x-axis in our layout
		var y_axisLength = 200; // length of y-axis in our layout
```

```
                    // Use a scale for the height of the visualization
                    var yScale = d3.scaleLinear()
                      .domain([0, maxValue])
                      .range([0, y_axisLength]);

                    //Create SVG element
                    var svg = d3.select("body")
                              .append("svg")
                              .attr("width", w)
                              .attr("height", h);

                    // Select and generate rectangle elements
                    svg.selectAll( "rect" )
                              .data( popData )
                              .enter()
                              .append("rect")
                              .attr( "x", function(d,i){
                                        return i * (x_axisLength/arrayLength) + 30; // Set x coordinate of rectangle to
index of data value (i) *25
                              })
                              .attr( "y", function(d){
                                        return h - yScale(d.pop2020) - 75; // Set y coordinate of rect using the y scale
                              })
                              .attr( "width", (x_axisLength/arrayLength) - 1)
                              .attr( "height", function(d){
                                        return yScale(d.pop2020); // Set height of using the scale
                              })
                              .attr( "fill", "steelblue");

                    // Create y-axis
                    svg.append("line")
                              .attr("x1", 30)
                              .attr("y1", 40)
                              .attr("x2", 30)
                              .attr("y2", 275)
                              .attr("stroke-width", 2)
                              .attr("stroke", "black");

                    // Create x-axis
                    svg.append("line")
                              .attr("x1", 30)
                              .attr("y1", 275)
                              .attr("x2", 250)
                              .attr("y2", 275)
                              .attr("stroke-width", 2)
                              .attr("stroke", "black");

                    // y-axis label
                    svg.append("text")
                              .attr("class", "y label")
                              .attr("text-anchor", "end")
                              .text("City Population")
                              .attr("transform", "translate(20, 40) rotate(-90)")
                              .attr("font-size", "14")
                              .attr("font-family", "'Open Sans', sans-serif");
          };


</script>
</body>
</html>
```

This example shows that D3 can be rather time consuming to achieve basic charts. When you are only creating basic line charts and bar charts then it is probably easier to use chart libraries such as Dimple.js and Plot.ly that speed up the process and reduce the length of code required.

D3 is more advantageous when you are looking to develop more sophisticated charts. An example has been included of a bubble chart using the force packing algorithm built into D3 in example 5-D3_example9_bubbles.html. This example is based on the template provided here- https://www.d3-graph-gallery.com/circularpacking.html

The template has been changed to use the city population data used in this practical. There are a number of ways this example could be improved on, such as clustering groups of cities by countries, and adjusting how the population variable corresponds to the circle size (at the moment it directly relates to radius, which is arguably misleading for the viewer, as it exaggerates differences between cities).



Further D3 Resources

There are many further examples of Mapbox GL functionality online to explore, on the main example pages and through various blogs that showcase recent developments in interactive mapping-

D3 tutorials in LinkedIn Learning- https://www.linkedin.com/learning/search?keywords=d3.js&u=69919578

D3 API Reference- https://github.com/d3/d3/blob/master/API.md

D3 Tutorials links- https://github.com/d3/d3/wiki/Tutorials

D3 Chart templates from the Financial Times- https://github.com/ft-interactive/visual-vocabulary