CASA0029: Urban Data 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 most popular of these in this practical: Chart.js. 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 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.

Importing CSV Data for Online Charts

The example HTML files and data for this practical are on Moodle. As with the online mapping examples from last week, 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 is typically 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 5).

For this practical we are going to use data from a CSV file (CityData_WUP2025.csv). This is a dataset of the largest urban agglomerations in the world according to the United Nations World Urbanization Prospects 2025. There are three versions of this CSV included- all 3000 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").then( function(CityData) {…} )
```

D3 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.

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.

b_CSVdata_example2.html

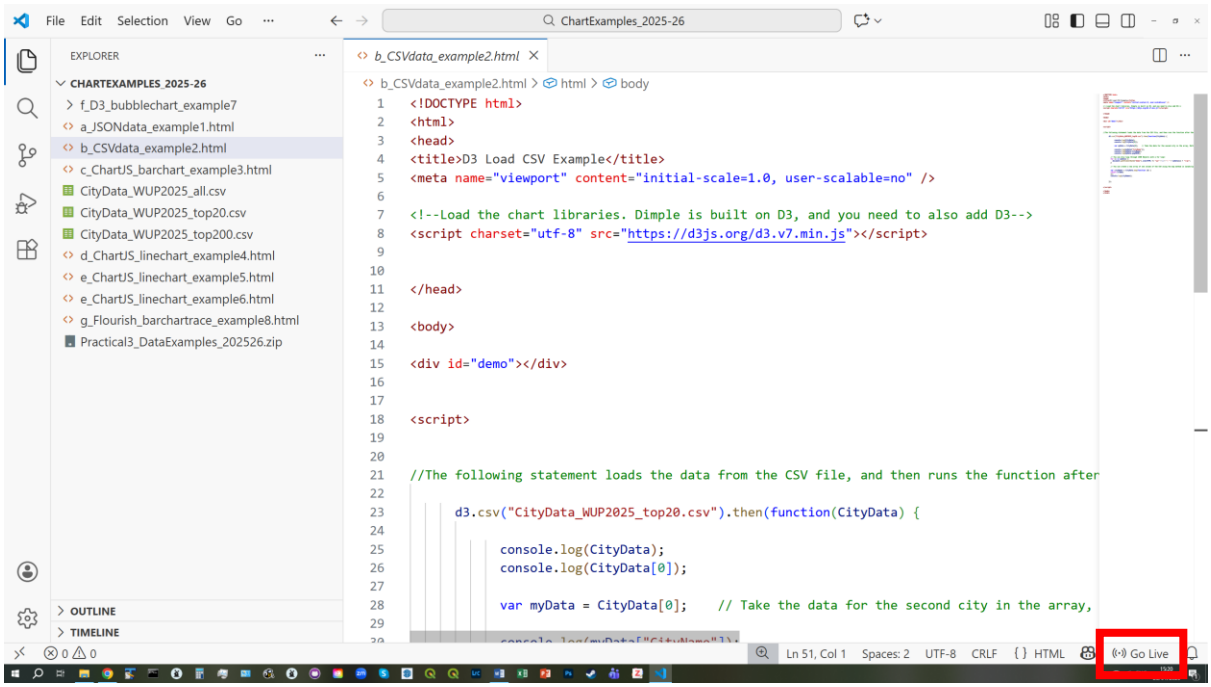
```
<!DOCTYPE html>
<html>
<head>
<title>D3 Load CSV Example</title>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

<!--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.v7.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_WUP2025_top20.csv").then(function(CityData) {

            console.log(CityData);
            console.log(CityData[0]);

            var myData = CityData[0];    // 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>";
        }

        // You can create a new array of one column of the CSV using the map method in
JavaScript. This runs a function for every element in the array.

        var cityNames = CityData.map(function (d) {
        return d.Name;
        });
        console.log(cityNames);


      });
</script>

</body>
</html>
```

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 Server in VS Code

You can get around the CORS error by running a local host. To do this in VS Code, you need to make sure to install the extension called Live Server- https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer



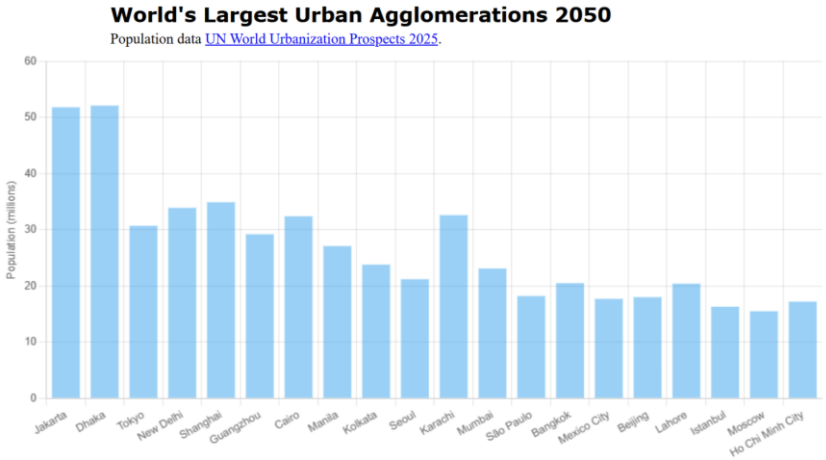
Uploading to a Web Server

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 in Week 5). Github Pages is a good free option for static websites that we will be using later in this module.

Creating a Bar Chart in Chart.js

Chart.js is a popular open source JavaScript charting library. It offers a series of standard chart types with relatively minimal coding required: https://www.chartjs.org/

Our first example is a Dimple bar chart, *Dimple_barchart_example3.html*. 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 Chart.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. Chart.js requires the labels and population data as an array of values. To achieve this we use the JavaScript .map function to extract the labels and 2050 population data as new arrays from the CityData array. These are assigned to the variables cityLabels and cityPop2050.

We then create a new chart of type bar, with cityLabels and cityPop2050 used as the data inputs-

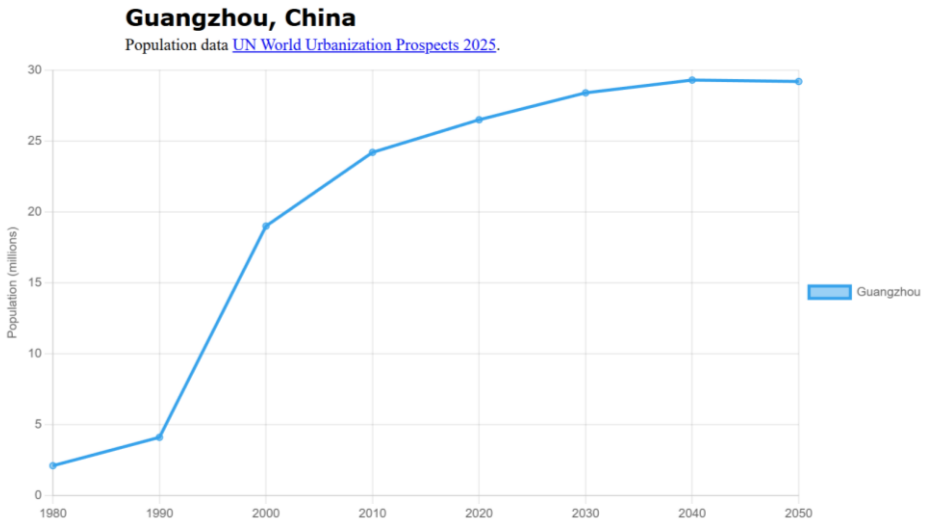
c_ChartJS_barchart_example3.html

```
<!DOCTYPE html>
<html>
<head>
<title>World's Largest Urban Agglomerations Bar Chart</title>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

<!--Load Chart.JS and D3 to parse the CSV file-->
<script charset="utf-8" src="https://d3js.org/d3.v7.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<style>

    #chartContainer {width: 900px; margin: auto; margin-top: 50px; }
    #chartContainer p {margin-left: 120px; 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 2050</p>
    <p id="chartsubhead">Population data <a
href="https://www.un.org/development/desa/pd/world-urbanization-prospects-2025">UN World
Urbanization Prospects 2025</a>.</p>
    <canvas id="myChart"></canvas>
</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_WUP2025_top20.csv").then(function(CityData) {

  const ctx = document.getElementById('myChart');

  var cityNames = CityData.map(function (d) {
  return d.Name;
  });

  var cityPop2050 = CityData.map(function (d) {
  return d.pop2050;
  });

  console.log(cityNames);
  console.log(cityPop2050);

  new Chart(ctx, {
    type: 'bar',
    data: {
    labels: cityNames,
    datasets: [
        {
        data: cityPop2050
        }
    ]
    },
    options: {
```

```
        plugins: {
            legend: {
                display: false
            }
        },
      scales: {
        y: {
          beginAtZero: true,
          title: {
            display: true,
            text: 'Population (millions)'
          }
        }
      }
    }
  });

});

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

Creating a Line Chart in Chart.js

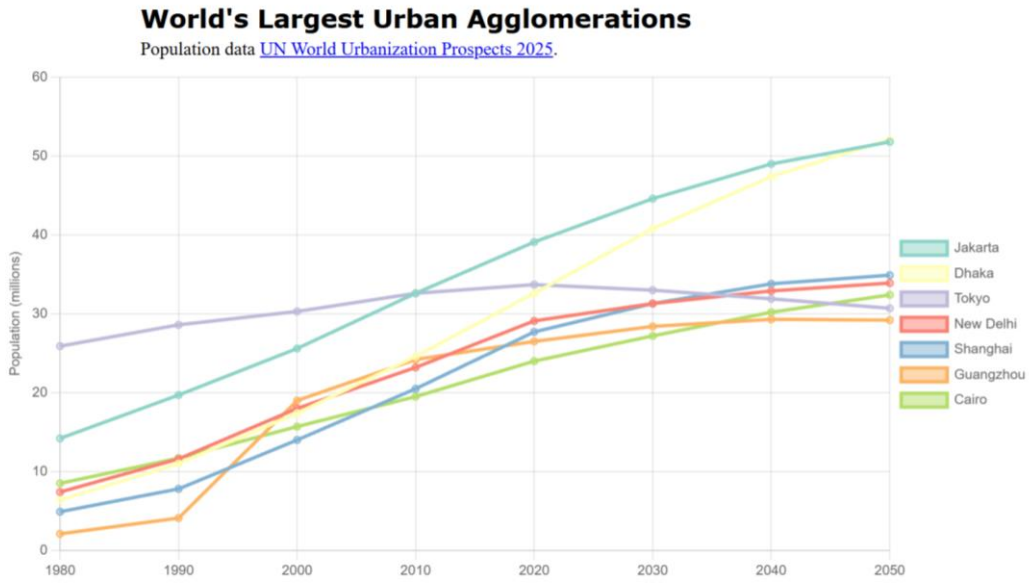
The next Chart.js example, d_ChartJS_linechart_example4.html, shows how to create a line chart using the same library-



This is very similar to the bar chart example. The main difference is that rather than using the .map method to create an array of data from parsed array from the D3 method, we simply create the array from a single city using the statement-

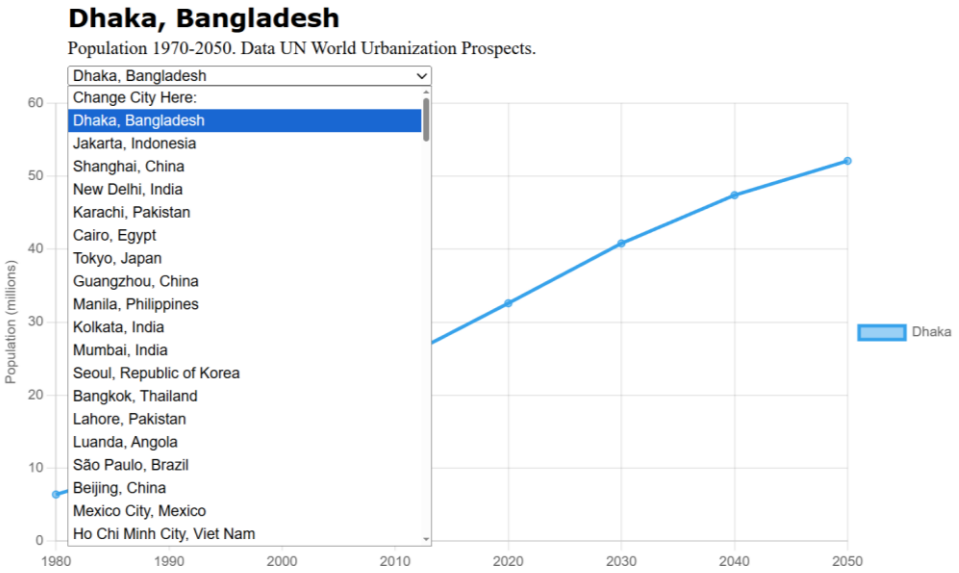
```
var PopulationData = [CityData.pop1980, CityData.pop1990, CityData.pop2000, CityData.pop2010,CityData.pop2020,CityData.pop2030,CityData.pop2040,CityData.pop2050];
```

Example 4 shows data for a single city, selected in the code (the variable CityDataIndex). Examples 5 and 6 explore different line chart approaches to mapping multiple cities. First of all Example 5 maps several cities on the same line chart-



In Example 5 rather than creating an array for a single city, a for loop is created to create a nested array of population data for multiple cities. Note this example loads a Chart.js plugin called colorschemes to improve the colour options available. The advantage here is that data for multiple cities is shown on a single chart, but there are legibility issues where more data is added.

Then Example 6 takes a different approach by providing a dropdown menu to the user, enabling the user to select the city of interest-



This approach avoids having too much data, but requires several clicks from the user to deliver the data they want. Generally designers advise showing the user the data as quickly as possible to help understanding.

D3 Example 7

Chart.js and similar libraries can achieve good quality basic charts relatively quickly, but do not have options for more specialist advanced charts. 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

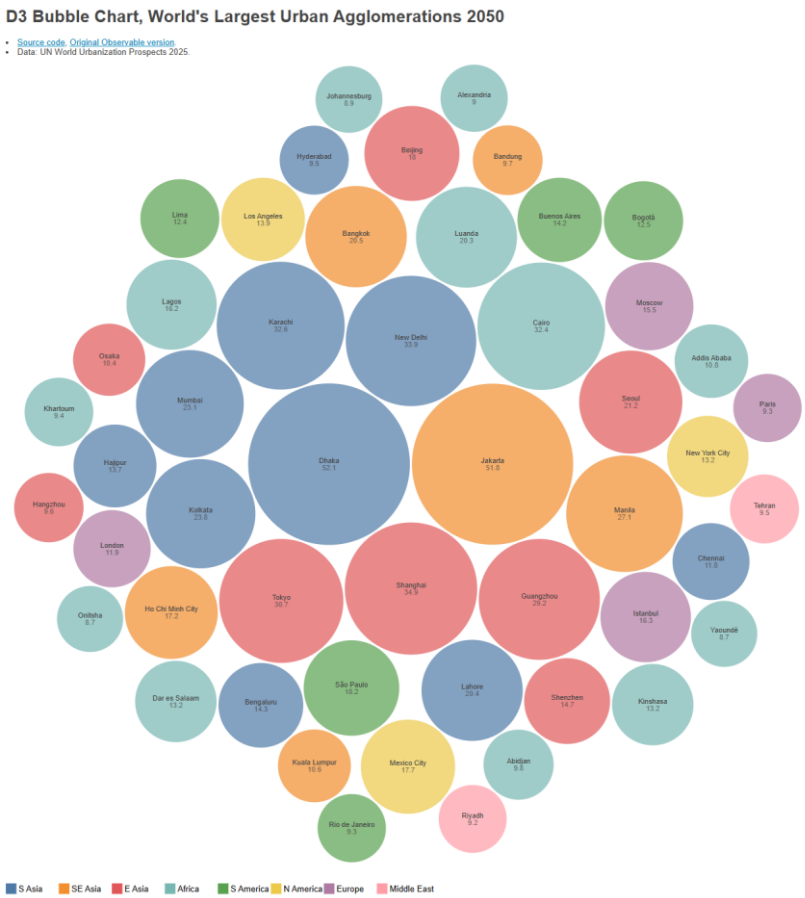
More recently, D3 has also developed an editable notebooks version for easy sharing of templates called Observable.

D3 is a powerful tool, though has quite a substantial learning curve and we will not go into detail on D3 in this module. If you want to develop your D3 skills, some useful tutorials can be found in the following locations-

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

D3 API Reference- https://d3js.org/api

Example 7 shows D3 being used to create a bubble chart of the world city population data, with colours showing the global region shown-



This example comes directly from the example in the Observable gallery, converted to standard JavaScript by Takanori Fujiwara.

Flourish Example

Learning a tool such as D3 can be time consuming. There are cloud service charting tools that can achieve similar outputs with much less effort. The downsides are that, similar to Mapbox, these are generally commercial providers, and free tiers can have limitations. They are also not typically open source in the same way as the previous tools shown in this practical.

Example 8 shows a fun animated visualisation created using the tool Flourish. This animated Bar Chart race focusses on the ranks of city populations changing over time using the same UN World Urbanization Prospects dataset-