**F21DV Lab 3 Report**

**Demonstrated to:** Benjamin Kenwright

**On date:** 25/03/2022

## Tech Used

The application is written using a combination of HTML, CSS and JavaScript using version 7 of the D3 library for data visualisation. To render the map I have used the Mapbox API and used D3 to visualise data on the map; this will be described in more detail in the Map section.

## Data Processing

I have used the prescribed COVID-19 dataset provided by ‘Our World in Data’. I initially read in the data from a csv file stored in the same directory as the main HTML file for the dashboard using the d3.csv(‘file\_name’) function.



One the function has read in the data as an array of objects it passes the result to the .then(function(data) { function where it is used in the creation of the dashboard.

### Grouping by month

My initial idea was to display on the map the number of deaths each month for each country, using circles where the radius represented the number of deaths. In order to do this I needed to sort the unordered dataset by month. To do this I initialised an empty object “monthGroup”. I then use a for loop to cycle through all objects in the data array, take their date property and convert it o a date object. From the date object I take the month and year and combine them into a string with the format “0 - 2020” for January 2020. This string is then used to look up the index of that month in the “months” object created before reading the data in. This object maps the “0 - 2020” to a human readable format and gives it an index so that the months can be stored ad displayed in order along the slider.

Graphical user interface, text, application

Description automatically generated

This index returned is then used as a key to create an object property in “monthGroup”, and the value is set to be an empty array. This creates a list of all the months in “monthGroup” and the values are empty arrays.

A screenshot of a computer

Description automatically generated with medium confidence

The next step is to populate the object with data corresponding to the correct month. I do this by using the same for loop but this time pushing the object into the array corresponding to the correct date.

Text

Description automatically generated

The result of this is an object where the keys are all the months the data covers and the values are arrays of all the data objects that correspond to that month.

A picture containing text

Description automatically generated

### Grouping by country

As the data will be shown by country, I created a function that will group the data given as an argument by country. Using the Array.prototype.reduce() function I loop through each item in the dataset. In the first iteration, an empty array is created and the array element is pushed to the empty array and a key value pair is created in “gourps” where the key is the country name of that array object. In the second and subsequent iterations, the “groups” object is passed in to the callback and either a new key:value pair is created for the new array object in that iteration or the array object is added to an existing array in the “groups” object if that country has already been encountered in the loop. The result is an object where the keys are the country names and the values are arrays of all the data objects corresponding to that country from the input data.

Text

Description automatically generated

A picture containing text

Description automatically generatedThis is used in the dashboard for the bar chart where all the data is grouped by country at once and in the map where a particular month of data is passed in and sorted by country.

### Coordinate data

For simplicity and speed, the data for each countries coordinates is stored as a data object in a separate JavaScript file (“newCoords.js”) and linked in the <head> section of the document. To get the coordinates, I used the Mapbox API forward Geocoding functionality which accepts a location name and returns its coordinates. I used the groupByCountry function to create a list of all countries and then iterated through the array, calling the API at each iteration to the return the coordinates. This took a while an interfered with the visualisations and so I felt it was easier to run it once and store the result.

Text

Description automatically generated

### Preparing data for visualisation

Once the data is grouped by country, the final data transformation takes place before being used in visualisation. The final data objects used by each visualisation contain the country name, the total number of deaths for that country in the input data (could be for a month or for all data), the number of people vaccinated, the country population, the gdp per capita, and the longitude and latitude of the country.

To do this, I use a for loop to loop through all of the countries in the input data object and then use a nested for loop to loop through all the data objects in that countries data array of the key:value pair. For each data object, the total deaths variable is incremented by the number of “new\_deaths” (if not null) in each data object. Then, if the country name is in the array of 16 countries being used “countrySelection” and object is created with all the values listed above.

The gdp\_per\_capita is taken from the first data object as it is fixed for all data objects for each country and the population is take from the last array element for the same reason. To get the number of people vaccinated, a for loop iterates backward through the array form the final element until it find a non-null value larger than 0. I found that sometimes in the last object of the array the number of people vaccinated was 0 so the loop finds the most recent vaccination figure. The latitude and longitude are then taken from the coordinates data object and the final object is pushed to the deaths array and returned.

Text

Description automatically generated