Student Declaration of Authorship

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

Course code and name:	F21DV Data Visualisation and Analytics
Type of assessment:	**Individual**
Coursework Title:	Lab 3
Student Name:	Josh Yang
Student ID Number:	091514042

Student Signature *(type your name):* *Josh Yang*

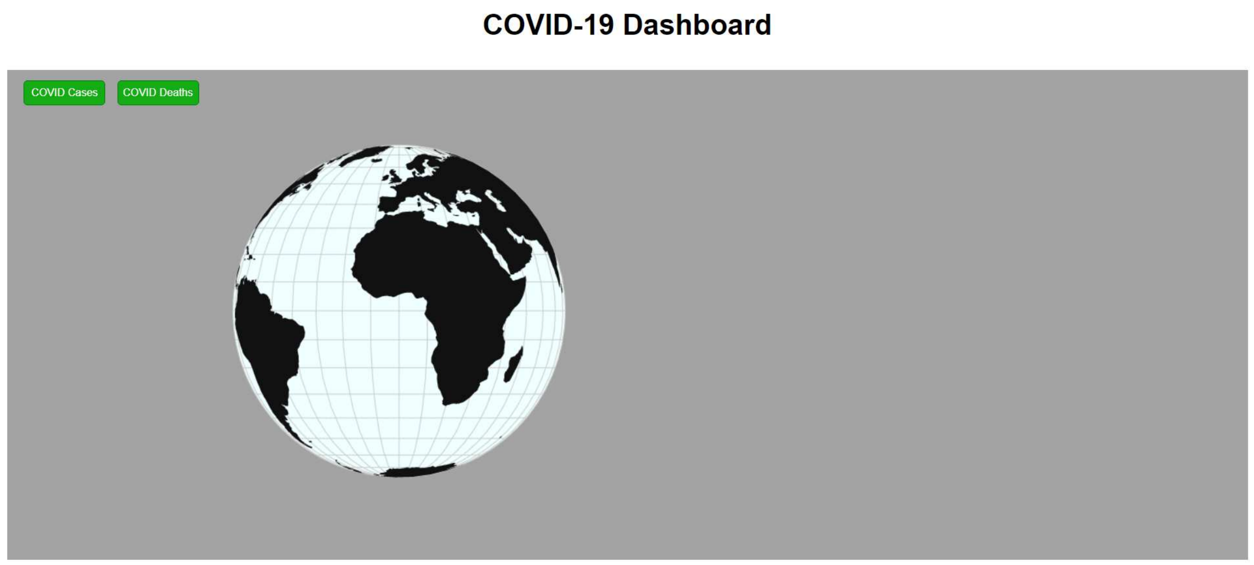
Date: *22/03/2022*

F21DV Data Visualisation and Analytics – Lab 3
Josh Yang jy84 091514042

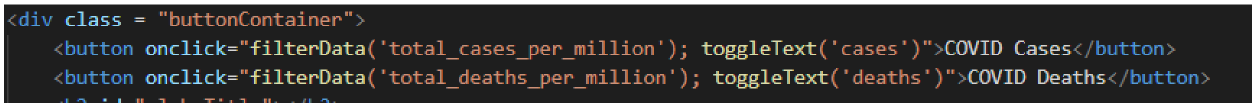
Demonstrated on 18/03/2022 to Amit Parekh.

The following report will provide further explanation and context on the complex, interactive visualisation solution developed based on the COVID-19 dataset provided.

I have committed all my answers to a public GitHub repository which can be accessed here:
https://github.com/JoshYang1/F21DV-Data-Visualisation-and-Analytics

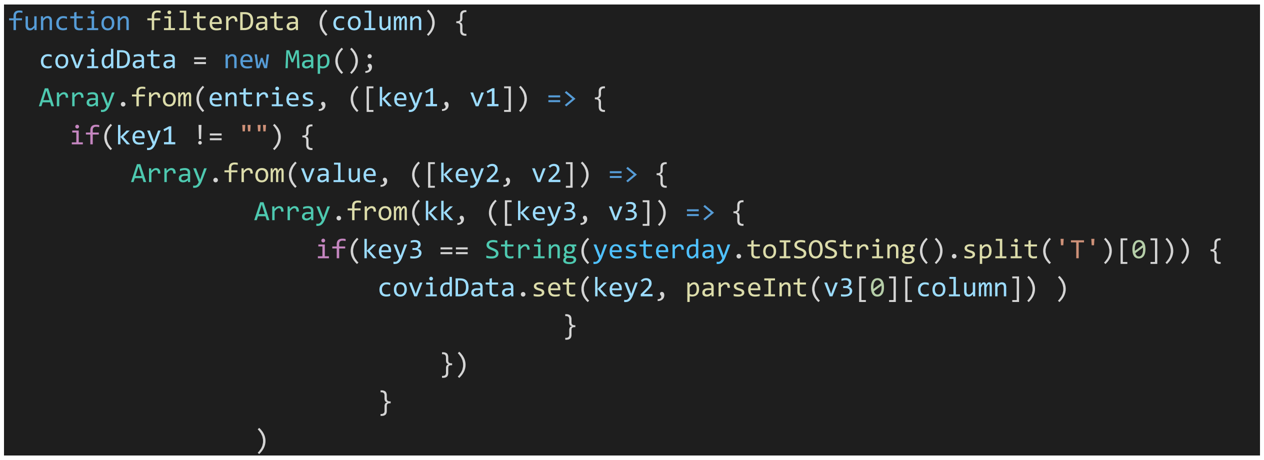


The above screenshot is the landing page for a user. There are two buttons in the top left which gives the user the functionality to pick between investigating COVID cases or COVID deaths.



```
<div class = "buttonContainer">
    <button onclick="filterData('total_cases_per_million'); toggleText('cases')">COVID Cases</button>
    <button onclick="filterData('total_deaths_per_million'); toggleText('deaths')">COVID Deaths</button>
```

Once either button is clicked it will call 2 functions to filter the data appropriately and to show the title, providing the user with feedback.



```
function filterData (column) {
  covidData = new Map();
  Array.from(entries, ([key1, v1]) => {
    if(key1 != "") {
        Array.from(value, ([key2, v2]) => {
                Array.from(kk, ([key3, v3]) => {
                    if(key3 == String(yesterday.toISOString().split('T')[0])) {
                        covidData.set(key2, parseInt(v3[0][column]) )
                            }
                        })
                    }
                )
```

```
            }
        })

      //https://stackoverflow.com/questions/48707227/how-to-filter-a-javascript-
map
      const max = new Map([...covidData].filter(([k,v]) => !isNaN(v) ));

      domain = [0,(Math.max(...max.values()))]

      colorScale = d3.scaleSequential(d3.interpolateViridis).domain(domain)

      legend();
      };
```

The first function, filterData, will create a new Map structure and then filter the dataset based on the parameter. We are only concerned with the latest data so we need to filter is for yesterday's date. We then removed all the entries that are NaN so we can set the domain accordingly. The legend function is then called so it can be displayed.

```
// reading the csv file
d3.csv(dataset).then(function(data) {

    // group the data
    // https://observablehq.com/@d3/d3-group-d3-hierarchy
    // https://observablehq.com/@d3/d3-hierarchy
    entries = d3.group(data, d => d.continent, d => d.location, d => d.date)

    startGlobe();
});
```

The dataset is read in and grouped by continent, country then date.

```
function startGlobe() {

  canvas.call(d3.drag()
      .on('start',  e => dragstarted(d3.pointer(e)) )
      .on('drag', e => dragged(d3.pointer(e)))
      .on('end', dragended)
    )
    .on('click', e => mousemove(d3.pointer(e)) )

  loadData(function(world, cList) {

      land = topojson.feature(world, world.objects.land)
      countries = topojson.feature(world, world.objects.countries)
      countryList = cList
      window.setInterval(update, 50);
```

Once the data has been read in and grouped, the startGlobe function is called. This will setup the interactivity of the globe and will create the polygons necessary to show the land and countries on the globe. The globe can be dragged so a user can find the country that they would like to investigate further. Once they have found the desired country, a user would then click on it.

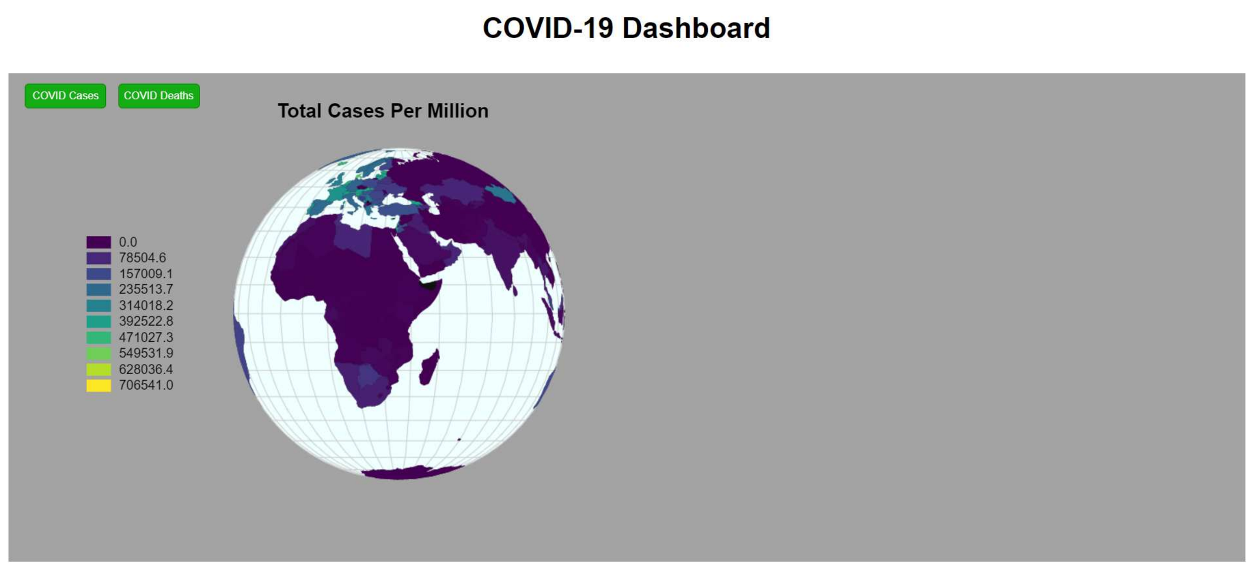
```
function mousemove(event) {
    var c = getCountry(event)
    if (!c) {
      if (currentCountry) {
        leave(currentCountry)
        currentCountry = undefined
        update()
      }
      return
    }
    if (c === currentCountry) {
      return
    }
    currentCountry = c
    update()
    enter(c)
  }
```

The mouse event details are passed into this function and we retrieve the country via the getCountry function. If a user clicks on anywhere except a country on the globe, the variable currentCountry is set to undefined and then the country previously selected is no longer highlighted.

```
function update() {

  context.clearRect(0, 0, 800, 600);
    fill({type: 'Sphere'}, '#F0FFFF')
    stroke(graticule, '#ccc')
    fill(land, '#111')

  attachData();

  if (currentCountry) {
    fill(currentCountry, '#a00')
  }
}

function fill(obj, color) {
    context.beginPath()
    geoGenerator(obj)
    context.fillStyle = color
    context.fill()
  }

  function stroke(obj, color) {
    context.beginPath()
    geoGenerator(obj)
    context.strokeStyle = color
    context.stroke()
  }
```

The update function will create a rectangle with the given width and height. Shapes are then passed to the fill function which are provided to the geoGenerator variable. The geographic path generator, d3.geoPath, given a GeoJSON geometry or feature object, renders the path to the Canvas element.

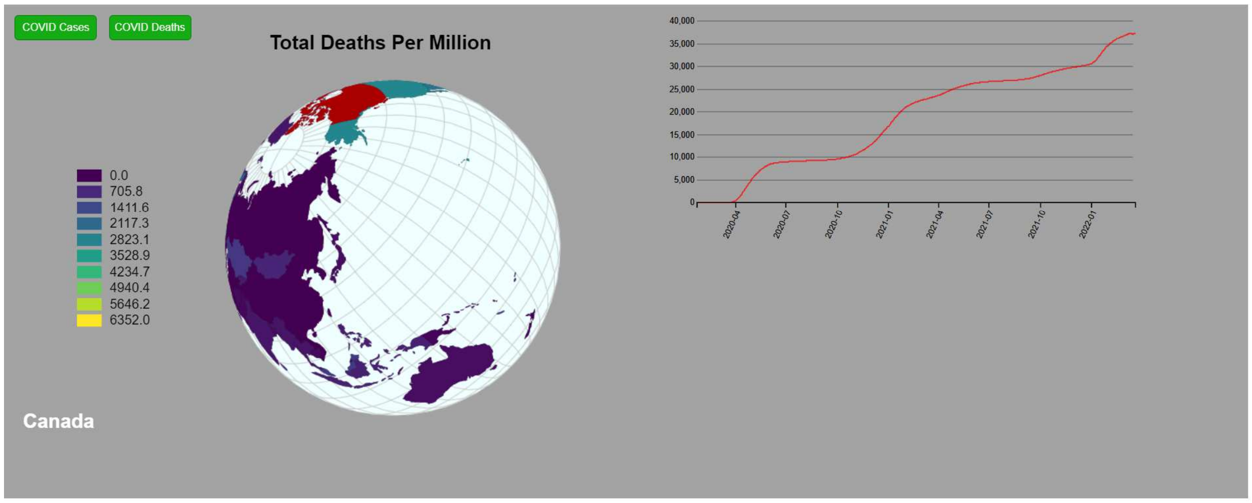
```
var context = canvas.node()
                    .getContext('2d');

let projection = d3.geoOrthographic()
                    .scale(width / Math.PI);

let geoGenerator = d3.geoPath()
                    .projection(projection)
                    .pointRadius(4)
                    .context(context);

let graticule = d3.geoGraticule10();
```



When either the COVID cases or COVID deaths button is clicked, each country is coloured based on the number of either parameter by per million population. The legend is shown providing information on the intervals and what each colour represents, from navy to yellow.

```
  function attachData() {

    if (covidData != undefined) {
    var ID
    var total
    var country

    // https://stackoverflow.com/questions/14379274/how-to-iterate-over-a-
javascript-object
    for (let key in countryList) {
        total = covidData.get(countryList[key].name) || 0;
        if (parseInt(countryList[key].id) < 0) {
          ID = countryList[key].id.replace('-','-00');
        } else if (parseInt(countryList[key].id) > 0 &&
parseInt(countryList[key].id) < 10) {
          ID = "00" + countryList[key].id;
        } else if (parseInt(countryList[key].id) >= 10 &&
parseInt(countryList[key].id) < 100) {
          ID = "0" + countryList[key].id;
        } else {
          ID = countryList[key].id
        }

        country = countries.features.find(o => o.id === ID)

        if (country != undefined) {
          fill(country, colorScale(total))
        }
      }
    }
  };
```

Firstly, we check that the button has been clicked so that the filtered data is available. Then for each key in the countryList, we retrieve the count from the filtered data or return 0. We then need to find the polygons with the country ID, so we need to do some String checking so that the IDs between the two datasets match. Finally, if we are able to determine the coordinates, the country is coloured based on the total number of cases / deaths.



When a country is selected, it will highlight red. See above which displays the globe on a slight tilt and will continue to rotate from that angle. Also, a line graph now appears to the right with data on that country. Additional feedback is provided in the bottom displaying the country selected name.

```
function enter(country) {
  var country = countryList.find(function(c) {
      return parseInt(c.id, 10) === parseInt(country.id, 10)
  })
  current.text(country && country.name || '')
  setLineData(country)
  };
```

The enter function will handle the display of the country name and provide the filter function setLineData with the same.

```
function setLineData(country) {
  if (country != undefined) {
    Array.from(entries, ([keyL1, valueL1]) => {
      if(keyL1 != "") {
        Array.from(valueL1, ([keyL2, valueL2]) => {
          if(keyL2 === country.name) {
            loadLineGraph(filterCountryData(valueL2));
          }
        })
      }
    })
  }
};
```

We check first that the country provided is defined. If so, we then filter the original grouped dataset, entries, checking that the continent information is available and then locating the country.

```
function filterCountryData(data) {
  var filteredMap = new Map();
  for (let key of data.keys()) {
    var obj = {};
    if (selection === "deaths") {
      death_columns.forEach(i => {
        obj[i] = data.get(key)[0][i]
      })
    } else if (selection === "cases") {
      cases_columns.forEach(i => {
        obj[i] = data.get(key)[0][i]
      })
    }

    filteredMap.set(key, obj);
  }
  return filteredMap;
}
```

We then pass in the filtered data to the filterCountryData function. Based on the variable selection (which is updated on the button click), the data is then further filtered based on the columns array. These columns are necessary for the line graph, and we wanted to reduce the amount of data passed between functions.

```
const death_columns = ["continent", "location", "date", "total_deaths"];
const cases_columns = ["continent", "location", "date", "total_cases"];
```

In the loadLineGraph function, we need to parse the data so that is in the necessary format for the visualisation.

```
  const lineData = [];

  if (selection === "deaths") {
    data.forEach(function(d) {
      var dt = parseTime(d.date);
      var v = parseInt(d['total_deaths']) || 0
      lineData.push({date: dt, value: v});
    })
  } else if (selection === "cases") {
    data.forEach(function(d) {
      var dt = parseTime(d.date);
      var v = parseInt(d['total_cases']) || 0
      lineData.push({date: dt, value: v});
    })
  }
```

```
  var country = svg.selectAll(".country")
```

```
                  .data([lineData]);

country.exit().remove();

country.enter().insert("g", ".focus").append("path")
        .attr("class", "line country")
        .style("stroke", "red")
        .merge(country)
        .transition().duration(1000)
        .attr("d", d=> line(d));
```

The data is loaded into the country variable which adds to the svg the .country element. The existing element is removed and a new one entered. Merge function will ensure there is a smooth transition between the different countries.

```
var line = d3.line()
  .curve(d3.curveCardinal)
  .x(d => x(d.date))
  .y(d => y(d.value));
```

```
  // Add X axis --> it is a date format
const x = d3.scaleTime()
            .rangeRound([0, width/1.2])
            .domain(d3.extent(lineData, function(d) {
              return d.date;
            }))

  // Add Y axis
const y = d3.scaleLinear()
            .rangeRound([ height/2, 0 ])
            .domain([0, d3.max(lineData, function(d) {
              return d.value;
            })])
            .nice();
```

```
svg.selectAll(".x-axis")
     .transition()
     .duration(1000)
     .call(d3.axisBottom(x)
             .tickFormat(d3.timeFormat("%Y-%m")))
     .selectAll("text")
     .style("text-anchor", "end")
     .attr("dx", "-.8em")
     .attr("dy", ".15em")
     .attr("transform", "rotate(-65)");
```

```
svg.selectAll(".y-axis")
    .transition()
    .duration(1000)
    .call(d3.axisLeft(y).tickSize(-width + margin.right + margin.left))
```

The x and y axis domain are based on the country dataset that is provided thus if a new country is selected the domain will also change.

We were not able to finish the tooltip and lineMouseMove functions before the deadline, unfortunately.