Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F21DV Lab 3
Type of assessment:	**Individual** *(delete as appropriate)*
Coursework Title:	Data Visualisation and Analytics
Student Name:	Jonathan Lee
Student ID Number:	H00255553

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Jonathan Lee*

Date: *01/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

F21DV Coursework Lab 2 Report

Jonathan Song Yang, Lee (H00255553)

Demonstrated to: Dr. Benjamin Kenwright (25/05/2022)

Contents

1 Introduction

Lab 3 focuses on using the power of data, and `d3.js` to build a covid-19 dashboard to convey a story about covid-19 using the provided covid-19 data.

Github Repo: https://github.com/jonleesy/F21DV-Coursework

Github Pages: https://jonleesy.github.io/F21DV-Coursework/public

1.1 Set-up

This dashboard uses a css-grid layout, as shown in listing 1.1 to tile the different objects on page. The program then uses a generalised function as shown in listing 1.2 to add div's to the grid layout.

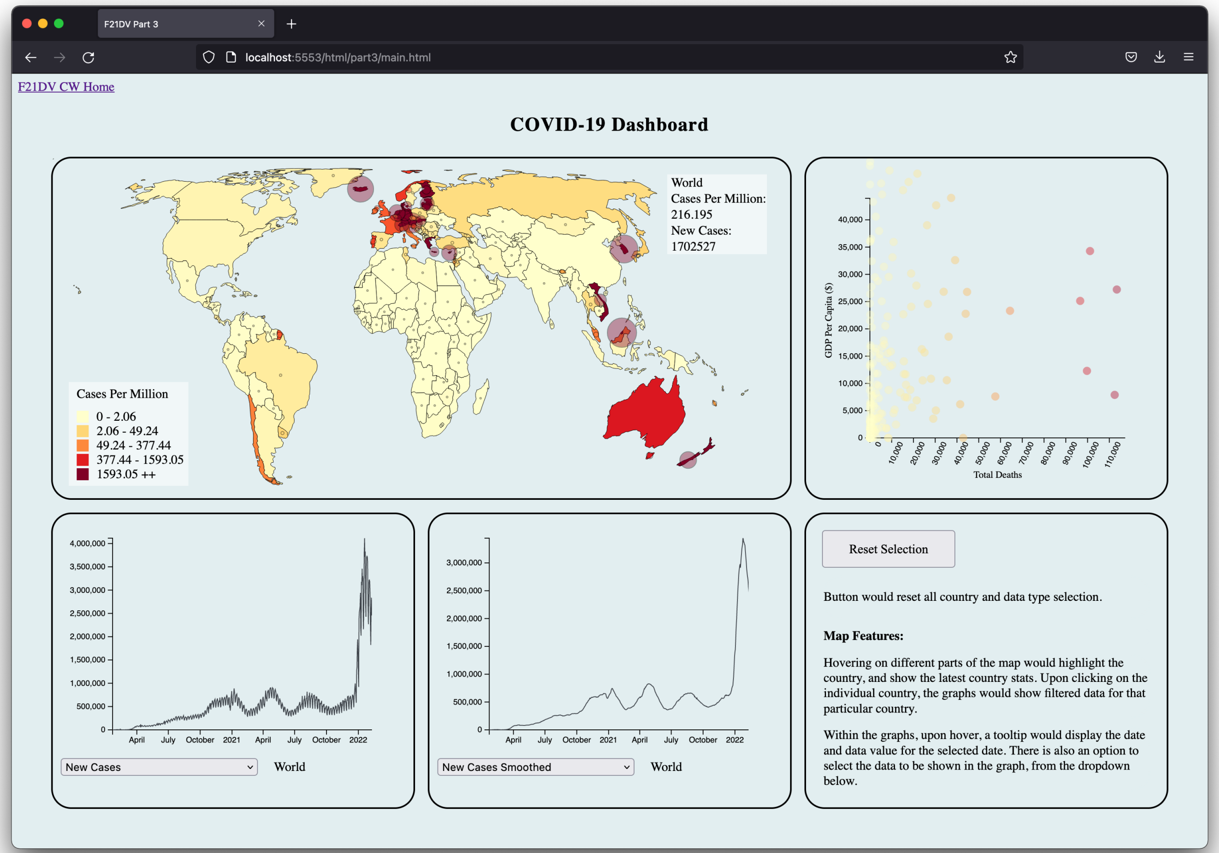
```css
.grid-container {
    width: 95%;
    max-width: 1400px;
    margin: 0 auto;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 1em;
}
```

Listing 1.1: css abstract

```javascript
export function createDivs(selector, className = '', idName = '') {
    d3.select(`.${selector}`)
        .append('div')
            .attr('class', className)
            .attr('id', idName)
}
```

Listing 1.2: generalised div creation function

2 The Application



World
Cases Per Million:
216.195
New Cases:
1702527

Button would reset all country and data type selection.

Map Features:

Hovering on different parts of the map would highlight the country, and show the latest country stats. Upon clicking on the individual country, the graphs would show filtered data for that particular country.

Within the graphs, upon hover, a tooltip would display the date and data value for the selected date. There is also an option to select the data to be shown in the graph, from the dropdown below.

Figure 2.1: Full Dashboard View

Shown in figure 2.1 is the main view of the Covid-19 Dashboard, made out of a few different layouts namely the map, the line graphs below, and a scatter plot.

2.1 The Map

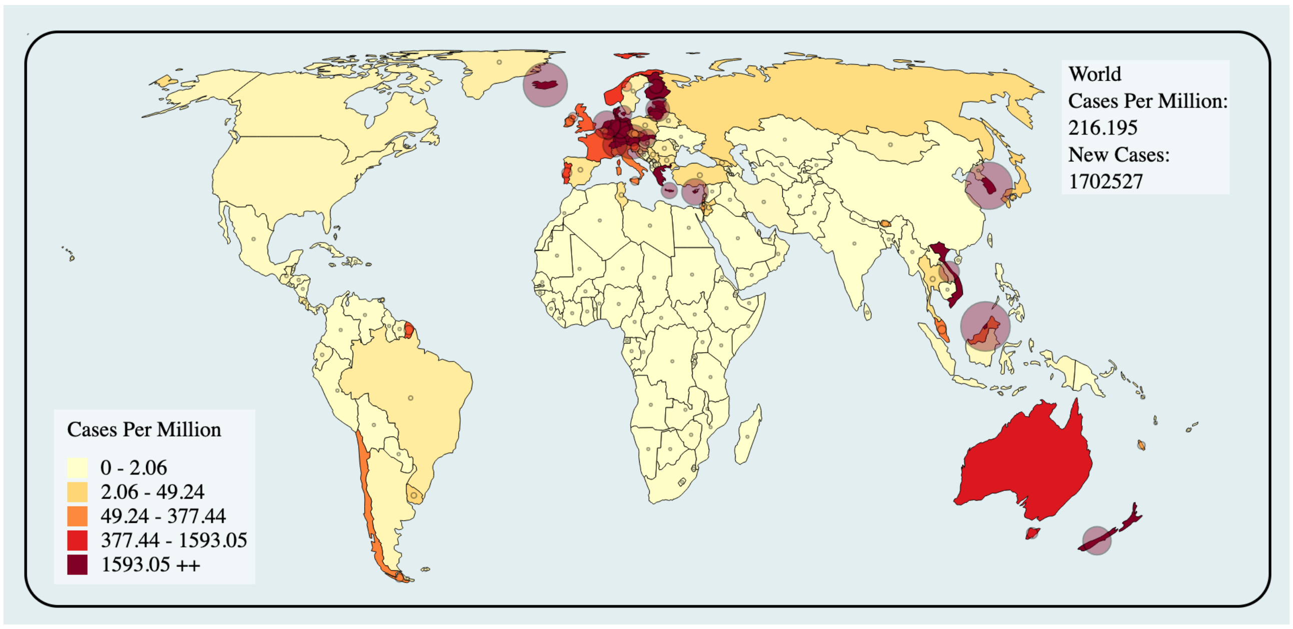


Figure 2.2: Closeup View of Map

A closeup view of the map, as shown in figure 2.2, would show a colour coded map based on the latest **New Cases Per Million** Covid-19 numbers, as well as circles indication where the covid hotspots all over the world are. The map also containes a legend on the top right showing a country's latest cases numbers upon mouse hover (World's data if mouse is not on any country).

The map container is first filled with an svg. Then the rest of the elements are just shapes and objects that were later on added onto the svg.

```
    // Use a projection to convert 'Coordinates' to 'pixels'.
    const mapProjection = d3.geoEqualEarth()
                             .scale(projectionScale)
                             // Adjust for translated top-margin in svg.
                             .center([0, mapMargin.top])
                             // Adjust for translated left-margin in svg.
                             .translate([mapWidth / 2 - mapMargin.left - mapMargin.
                                 ↪ right,
                                           mapHeight / 2]);
```

Listing 2.1: Map Projection

From listing 2.1, we could see the projection that was used to scale the geojson polygons and add them to the map. I have chosen the `.geoEqualEarth()` scaling method since it is the best way to see the size of different countries in relation to one another. This would be the best choice for viewing covid data, as we could now see how big and small countries react to covid-19 differently.

```
    // Plot the map and colours
    mapSvg.append('g')
            .selectAll('path')
            .data(topo[0].features)
            .enter()
                .append("path")
                .attr('d', d3.geoPath().projection(mapProjection))
```

```
8                    // fill colours
9                    .attr('fill', d => {
10
11                       // Use data's id to match and find the number of cases
12                       let cases = dataCase.find(data => data[0] == d.id);
13
14                       // Process NaN
15                       let total = cases ? cases[1] : 0;
16                       if (total == 'no data') {
17                           total = 0;
18                       }
19
20                       // Return the colours.
21                       return mapColourScale(parseFloat(total));
22                   })
23                   .attr('stroke', 'black')
24                   .attr('stroke-width', '0.5px')
25                   .attr('class', 'map')
26               .on('mouseover', mouseOver)
27               .on('mouseleave', mouseLeave)
28               .on('click', click)
```

Listing 2.2: Adding the Map

Listing 2.2 shows the how the map is added to the svg. `topo[0].features` is the geojson file that was passed into the d3 function. An abstract of it can be seen in appendix section 4.1. Using `d3.geopath()`, the map could be added onto the svg, country by country, and the fill of each country — the colour coded case numbers of each country is added on using the function in line 9 to 20.

The function first takes the data, `d` value for each country and obtain its id, `d.id`, then uses a default JavsScript array function `.find()` to obtains the latest case numbers from the `dataCase` array, which stores the latest case numbers for all the countries. Then, a function that checks for null value is in place to ensure that even if a country does not have any numbers, they could still have a colour. Lastly, the case numbers are passed into the colour scale function to return a colour value. The same thing is also for the circles, just that there is also a function that sizes the circles according to the case numbers.

2.1.1 Map on Mouse Over

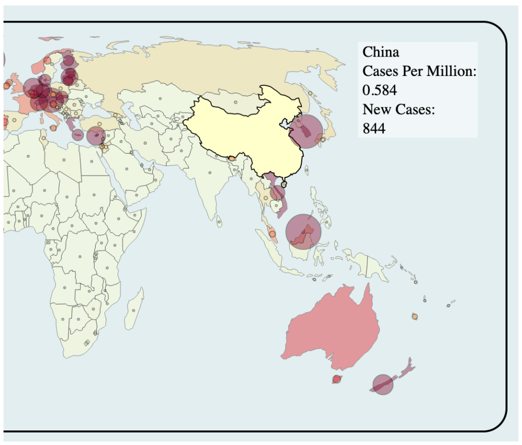


Figure 2.3: Map on Mouse Hover

Figure 2.3 shows what happens to the map on mouse over action. Upon mouse over, on any country, the map's upper right tooltip would show the country's name, cases per million, as well as the latest new cases numbers.

```
    // Mouse over function.
    function mouseOver(_, d) {
        const newTolltipData = dataCase.find(data => data[0] == d.id);
        const newCountry = newTolltipData[2];
        const newCasesPerMil = newTolltipData[1];
        const newCases = parseInt(newTolltipData[3]);

        // Blur the rest
        d3.selectAll('.map')
            .transition()
            .duration(transitionDuration)
            .style('opacity', 0.4)
            .attr('stroke-width', '0.5px')

        // Blur except the selected
        d3.select(this)
            .transition()
            .duration(transitionDuration)
            .style('opacity', 1.0)
            .attr('stroke-width', '1px')

        // Rename Tooltips
        d3.select('.tooltip-country')
            .text(newCountry)
        d3.select('.tooltip-cases-per-mil')
            .text(newCasesPerMil)
        d3.select('.tooltip-new-cases')
            .text(newCases)
    }
```

Listing 2.3: Map's Mouse Over Function

Figure 2.3 shows how the mouseover function is achieved. On mouseover, d3 parses the data through the function, and once again, using the native `.find()` function for JavaScript arrays, we find the data for said country, and using that, we rename the tooltip using a `d3.select()` and `.text()` function. The function also highlights the selected country, and increases its stroke width so that it is more obvious in highlight.

In this case, since the latest data is on the 14th of March, china has 0.584 cases per million people, and have 844 new cases on that day.

2.1.2 Map on Mouse Click

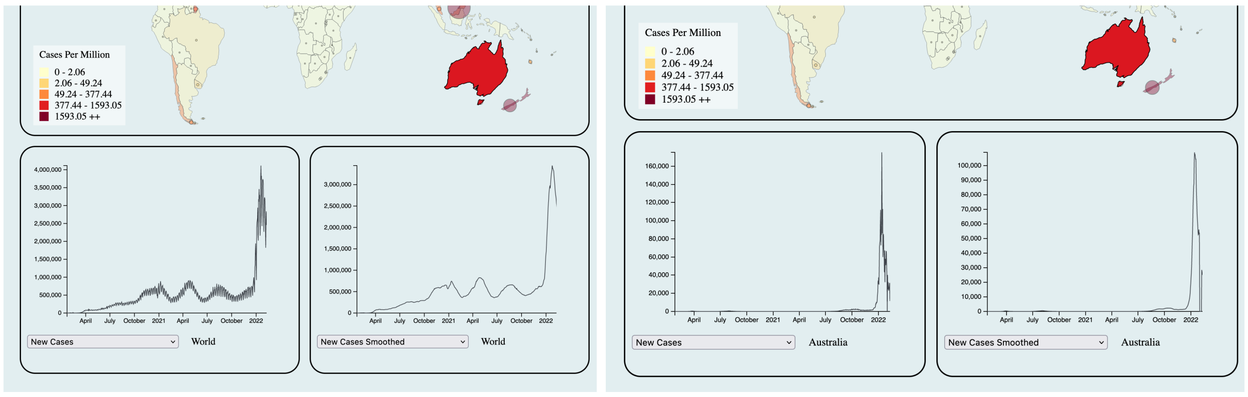


Figure 2.4: Map on Mouse Click

Figure 2.4 shows the before and after of clicking on **Australia** in the map. Upon clicking, the graph section below would be updated with the Australian case numbers, by default, it would be New Cases and New Cases Smoothed. The x and y axis would be updated too. The detailed update mehtod of the graph would be further dicussed later.

```
    // onClick function for each country.
    async function click(_, d) {

        // Get the country id.
        countryConst = d.id;

        // Get the data for this country.
        const thisdata = await getCountryData(d.id);

        // Filter to get country's name, to update name div.
        countryNameConst = thisdata.find(k => k.location);

        // Update the graphs.
        updateGraph(thisdata, typeConst, 'graph1');
        updateGraph(thisdata, typeConst2, 'graph2');

        // Check and show if no data.
        if (thisdata.length == 0) {
            d3.selectAll('.graph-container span')
                .text(`${d.properties['name']} **No Data`);
        } else {
            d3.selectAll('.graph-container span')
                .text(d.properties['name']);
        }
    }
```

Listing 2.4: Map's Mouse Click Function

Listing 2.4 shows the click function processing the mosuse click event on the map. Upon click, the mouse would obtain the country's data using a predefined function `getCountryData()` which returns an array of all the country's data. Using the same array, and the native `.find()` function, we then obtain the country's name. Then using the `updateGraph()` function, the graphs are now both updated. If the array containes no data, there would also be a no data sign that is shown beside the country name.

2.2 The Graph

```
    // Define the Axes
    const horScale = d3.scaleTime()
                        .domain(d3.extent(data.map(d => new Date(d.date))))
                        .range([0, graph1width]);
    const verScale = d3.scaleLinear()
                        .domain([0, d3.max(data.map(d => parseFloat(d[type])))])
                        .range([graph1height, 0]);
```

Listing 2.5: Graph Axes Scale

```
    // Define a line.
    const graph1line = d3.line()
                    .x(d => horScale(new Date(d.date)))
                    .y(d => verScale(d[type]))

    // Add the initial path.
    graph1svg.append('path')
            .attr('class', `${selector}line`)
            .attr('fill', 'none')
            .attr('stroke', 'black')
            .attr('d', graph1line(data))
```

Listing 2.6: Plotting the Graph

Listing 2.5 shows the scale method used for the X and Y axes of the graph. Using these scales, later then in 2.6, they were used to define the line for the graphs. Finally, to draw the base of the graph, these scales and lines were used to draw the base graph, before calling an update function.

2.2.1 The updateGraph() function

The function takes on `dataUpdate`, `type` and `selector` as parameters, where `dataUpdate` is the newest data — to be used with upon click of a country, `type` to be used with selecting the dropdown menu on the graph, and finally the `selector` to differentiate the use of said function of the different graph.

Upon update, the graphs do many things. Amongst which are replotting the graph, and renaming the country selected.

Updating the Graph Line

```
    // Define the Axes
    const horScale = d3.scaleTime()
                        .domain(datesExtent)
                        .range([0, graph1width]);
    const verScale = d3.scaleLinear()
                        .domain([0, d3.max(dataUpdate.map(d => parseFloat(d[type])))
                            ↪ ])
                        .range([graph1height, 0]);

    // Re-Call the Axes.
    d3.selectAll(`.${selector}bottomAxes`)
        .transition()
            .duration(400)
            .ease(d3.easeLinear)
            .call(d3.axisBottom(horScale))
    d3.selectAll(`.${selector}leftAxes`)
        .transition()
            .duration(400)
            .ease(d3.easeLinear)
            .call(d3.axisLeft(verScale))

    // Define a line.
    const graph1line = d3.line()
                        .x(d => horScale(new Date(d.date)))
                        .y(d => verScale((d[type])))

    // Update the svg line data.
    d3.selectAll(`.${selector}line`)
            .transition()
            .duration(400)
            .ease(d3.easeLinear)
            .attr('d', graph1line(dataUpdate))
            .attr('stroke', '#3e434a')
```

Listing 2.7: Updating the Graph

Listing 2.7 shows the code that was used to update the line graph. First, using the new data, `dataUpdate`, a new x and y axis scale was obtained, and then re-added onto the graph. A transition was also used, to smoothly add the axis.

Later, on line 22 onwards, a new line is defined, and then recalled onto the existing line using a transition.

Adding a Tooltip

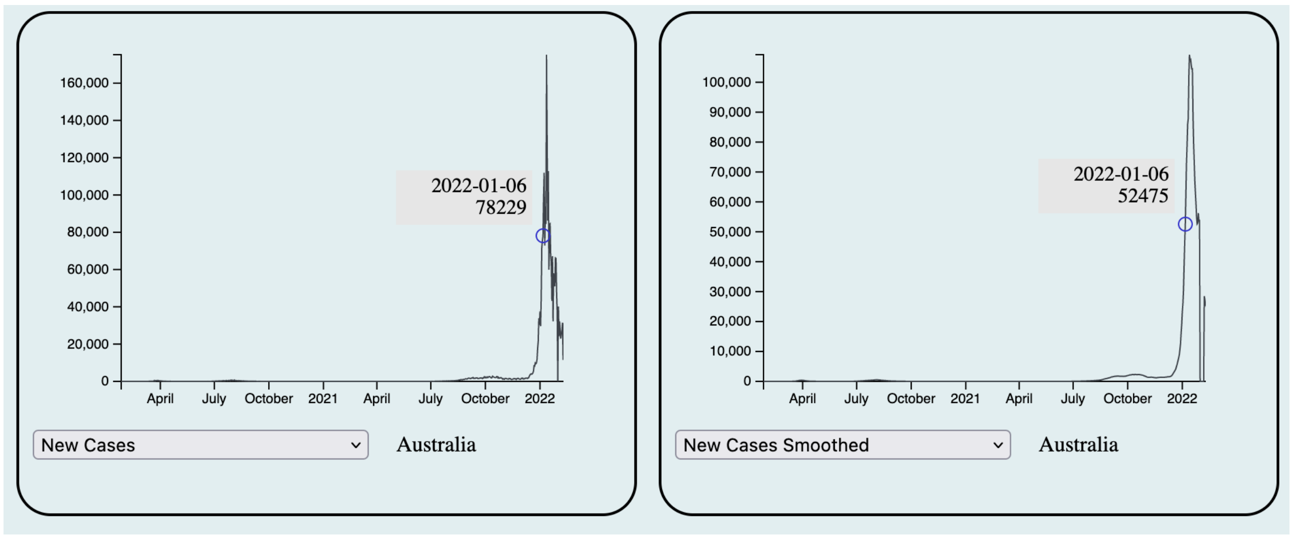


Figure 2.5: Map on Mouse Click

Figure 2.5 shows a tooltip appearing on hover of the line graph. Due to the length of the code, the code could be viewed here.The idea is that both graphs would sahre the same x-axis value, hence the same point on the x-axis. The only difference is the different 'type' of data. Hence, there was two difference type of y scale, hence a different translation of the y axis for both the graphs.

2.2.2 The Dropdown Selector

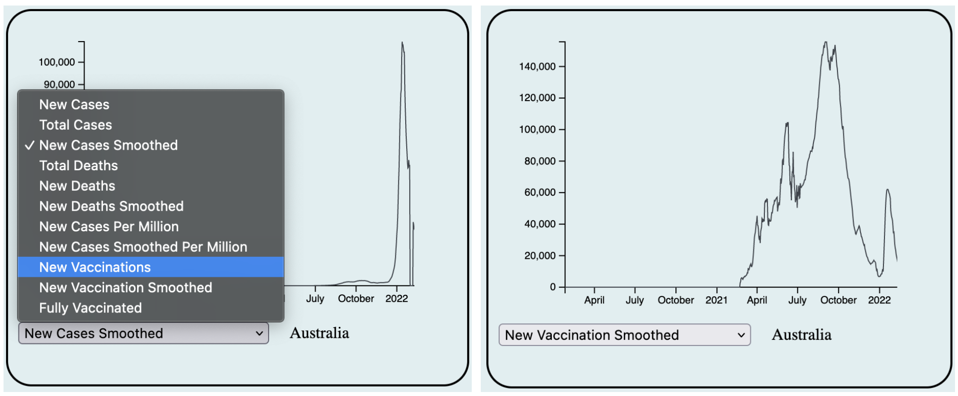


Figure 2.6: Graph Selector

Figure 2.6 shows the dropdown selector for each individual graph. From the figure shown, we could see that upon select, the graph would be updated with a new data. This would use the `updateGraph()` function as well, and updating the ‘type’ parameter.

2.3 The Scatter Plot

The scatter plot is just a simple case of adding circles to the svg. This is by using the latest New Deaths data, and the newest GDP per capital data to determin a circle’s X and Y axis. The circle is then colour-coded using the same colour scale for the map.

2.4 The reset button

The reset button does 2 things:

- Resets the graph selection to world’s data.
- Resers individual graph’s data to default: New Cases and New Cases Smoothed.

3 Queries

3.1 Is there a relation between the relative 'wealth' of a population and the evolution of the pandemic?

We could see from the scatter plot, that there is a cluster of low GDP Per Capita countries with low deaths, this is at the lower left of the graph. The rest of the countries are just scattered around the graph. This does not show anything about the relation between wealth and population. This just shows that there are just more countries with a low GDP, hence also a higher percentage of countries with low death rates.

3.2 What is the effect of vaccinations on the spread of cases/deaths and display any effect (if any) that booster jabs have on the cases/deaths.

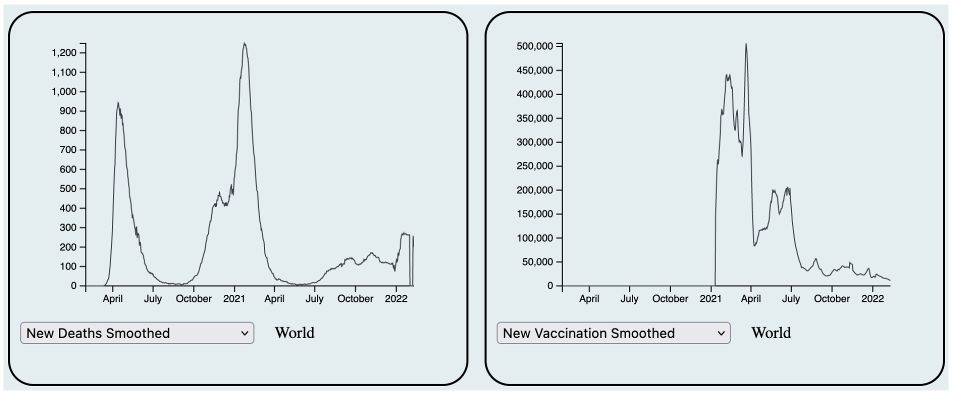


Figure 3.1: Vaccinations on death

Figure 3.1 shows the effect of vaccinations on the death rates. As we could see, as people started getting vaccinated, the peaking death rates started to come back down as well. To know if there is a full effect of vaccinations on the death rates, there needs to be some coorelation analysis on these two data.

4 Appendices

4.1 GeoJson File

```
{"type":"Feature","properties":{"name":"Azerbaijan"},"geometry":{"type":"
    ↪ MultiPolygon","coordinates"
    ↪ :[[[[45.001987,39.740004],[45.298145,39.471751],[45.739978,39.473999],[45.735379,39.3
    ↪ "id":"AZE"},
{"type":"Feature","properties":{"name":"Burundi"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[29.339998,-4.499983],[29.276384,-3.293907],[29.024926,-2.839258],[29.632176,-2.91
    ↪ "id":"BDI"},
{"type":"Feature","properties":{"name":"Belgium"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[3.314971,51.345781],[4.047071,51.267259],[4.973991,51.475024],[5.606976,51.037298
    ↪ "id":"BEL"},
{"type":"Feature","properties":{"name":"Benin"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[2.691702,6.258817],[1.865241,6.142158],[1.618951,6.832038],[1.664478,9.12859],[1.
    ↪ "id":"BEN"},
{"type":"Feature","properties":{"name":"Burkina Faso"},"geometry":{"type":"
    ↪ Polygon","coordinates"
    ↪ :[[[-2.827496,9.642461],[-3.511899,9.900326],[-3.980449,9.862344],[-4.330247,9.610835
    ↪ "id":"BFA"},
{"type":"Feature","properties":{"name":"Bangladesh"},"geometry":{"type":"Polygon"
    ↪ ,"coordinates"
    ↪ :[[[92.672721,22.041239],[92.652257,21.324048],[92.303234,21.475485],[92.368554,20.67
    ↪ "id":"BGD"},
{"type":"Feature","properties":{"name":"Bulgaria"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[22.65715,44.234923],[22.944832,43.823785],[23.332302,43.897011],[24.100679,43.741
    ↪ "id":"BGR"},
{"type":"Feature","properties":{"name":"The Bahamas"},"geometry":{"type":"
    ↪ MultiPolygon","coordinates"
    ↪ :[[[[-77.53466,23.75975],[-77.78,23.71],[-78.03405,24.28615],[-78.40848,24.57564],[-7
    ↪ "id":"BHS"},
{"type":"Feature","properties":{"name":"Bosnia and Herzegovina"},"geometry":{"
    ↪ type":"Polygon","coordinates"
    ↪ :[[[19.005486,44.860234],[19.36803,44.863],[19.11761,44.42307],[19.59976,44.03847],[1
    ↪ "id":"BIH"},
{"type":"Feature","properties":{"name":"Belarus"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[23.484128,53.912498],[24.450684,53.905702],[25.536354,54.282423],[25.768433,54.84
    ↪ "id":"BLR"},
{"type":"Feature","properties":{"name":"Belize"},"geometry":{"type":"Polygon","
    ↪ coordinates"
    ↪ :[[[-89.14308,17.808319],[-89.150909,17.955468],[-89.029857,18.001511],[-88.848344,17
    ↪ "id":"BLZ"},
```

Listing 4.1: GeoJson Abstract