DBMS DA214

Assignment 2

-By Rishita Agarwal

-220150016

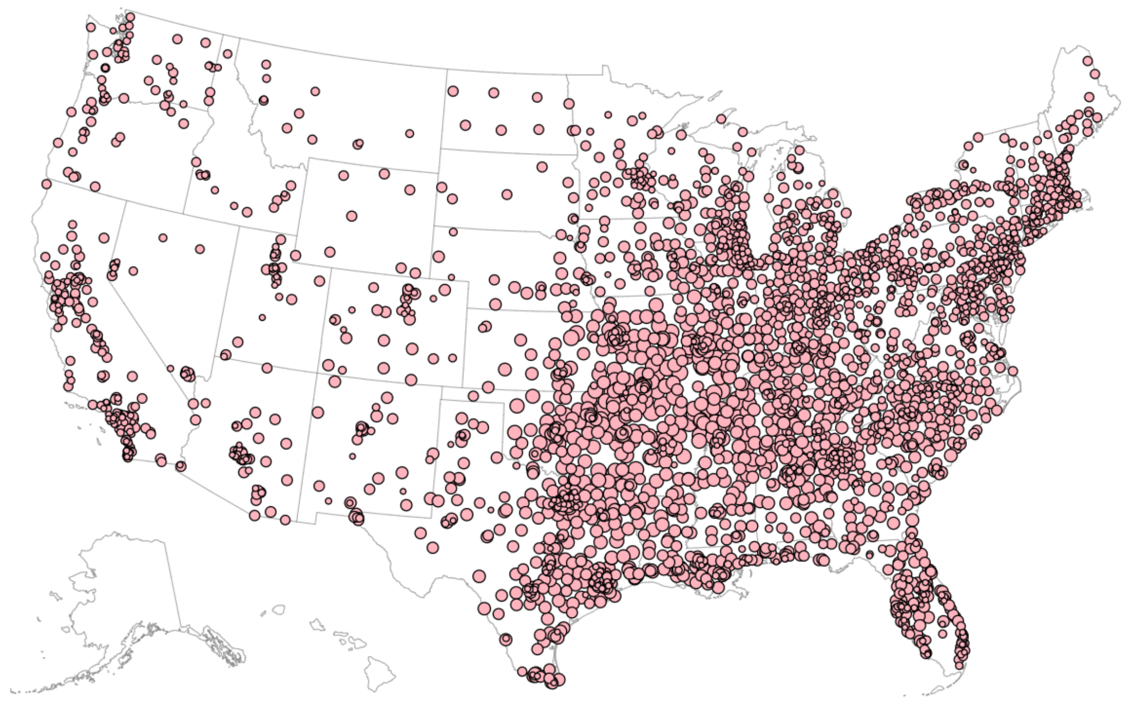
Hexbin map / D3 | Observable (observablehq.com)

Coronavirus (Covid-19) Data of United States (USA) (kaggle.com)

Github repository for all the codes and reports:
rishita3003/DBMS-d3.js-Assignments (github.com)

Question 1

The existing set has a dataset with geo-location and date of Establishment of that Walmart location. The task that is done is to plot the locations with bubbles, whose size is proportional to its age. Repeat the task on your local machine and note down the locations of the functions that do the task.



https://observablehq.com/d/d2e8b2ff2bb236dd

Link to the observable notebook is above.

Code (d3.js library of javascript)

```
chart = {

 // Specify the map’s dimensions and projection.

  const width = 928;

  const height = 581;

  const projection = d3.geoAlbersUsa().scale(4 / 3 * width).translate([width / 2, height / 2]);

  // Create the container SVG.

  const svg = d3.create("svg")

      .attr("viewBox", [0, 0, width, height])

      .attr("width", width)

      .attr("height", height)

      .attr("style", "max-width: 100%; height: auto;");

  // Append the state mesh.

  svg.append("path")

      .datum(stateMesh)

      .attr("fill", "none")

      .attr("stroke", "#777")

      .attr("stroke-width", 0.5)

      .attr("stroke-linejoin", "round")

      .attr("d", d3.geoPath(projection));
  svg.append("g")
  .selectAll("circle")
  .data(walmarts)
  .join("circle")
    .attr("cx",d=>projection([d.longitude,d.latitude])[0])
    .attr("cy",d=>projection([d.longitude,d.latitude])[1])
    .attr("r",d=>{
      const age = (new Date()).getFullYear()-(new Date(d.date)).getFullYear();
       return age*0.12;
    })
    .attr("fill","lightpink")
    .attr("stroke","#000")
  .append("title")
  .text(d=>`(${(new Date()).getFullYear() - (new Date(d.date)).getFullYear()}  years old)`);
  return svg.node();
}
```

Explanation : Only the chart is shown in the above code as rest of the code importing the walmarts and state mesh etc. are similar to the one given in the hexbin link in the assignment pdf.

How the code works:

Defined the geographic projection to display the US, with a scaling factor and centering translation based on the SVG size.

Created a new SVG element with a specified ‘viewBox’ and size, and sets the style for it to be responsive.

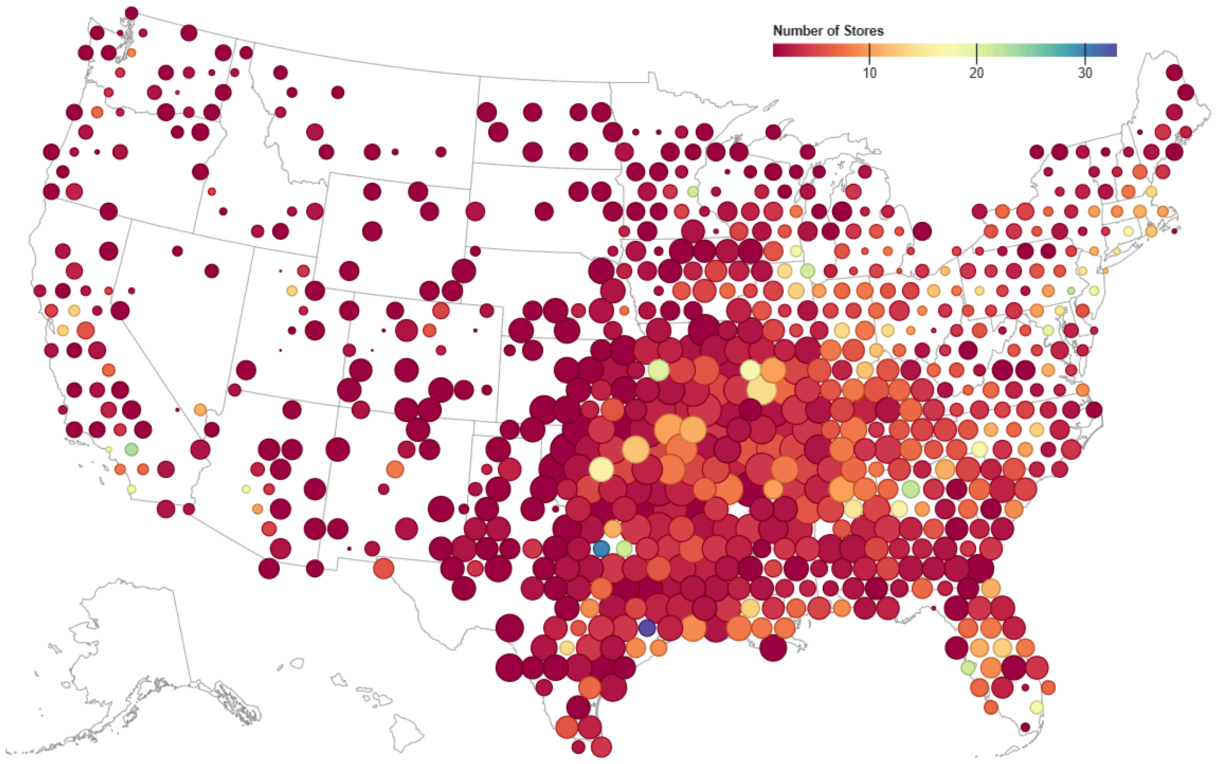
Drew state borders on the map using the path.

Circles are plotted on the US map with the radius on the basis of age of the stores, by calculating number of years since each store was established.

‘Title’ element to each circle, which acts as a tooltip displaying the age of the store when hovered over with a mouse.

Question 2

Can you repeat the above work, but now you are aggregating the summation/average age within a certain diameter and plotting the circles.



https://observablehq.com/d/45f3a0da474954ea

Link to the observable notebook is above.

Code (d3.js library of javascript)

```
chart = {

  // Specify the map's dimensions and projection.

  const width = 928;

  const height = 581;

  const projection = d3.geoAlbersUsa().scale(4 / 3 * width).translate([width / 2, height / 2]);

  // Create the container SVG.

  const svg = d3.create("svg")

      .attr("viewBox", [0, 0, width, height])

      .attr("width", width)

      .attr("height", height)

      .attr("style", "max-width: 100%; height: auto;");

  // Create the bins.

  const hexbin = d3.hexbin()

      .extent([[0, 0], [width, height]])

      .radius(10)

      .x(d => d.xy[0])

      .y(d => d.xy[1]);

  const bins = hexbin(walmarts.map(d => ({

    xy: projection([d.longitude, d.latitude]),

    date: new Date(d.date) // Convert the date string to a Date object
  }))) .map(d => {
    // Calculate the median age for the bin
    const meanAge = d3.mean(d, d => (new Date()).getFullYear() - d.date.getFullYear());
    return { length:d.length,x:d.x,y:d.y, meanAge:meanAge };
  })
  .sort((a, b) => b.meanAge - a.meanAge); // Sort bins by medianAge
   // Create the color scale.
  const color = d3.scaleSequential(d3.extent(bins, d => d.length), d3.interpolateSpectral);
  // Define the radius scale
  const radius = d3.scaleSqrt()
      .domain( d3.extent(bins, d => d.meanAge))
      .range([0, hexbin.radius() * Math.SQRT2]);
  svg.append("g")
      .attr("transform", "translate(580,20)")
      .append(()=>legend({
        color,
        title: "Number of Stores",
        width: 260
        // Define appropriate tick values and format for the legend
      }));
  // Append the state mesh.
```

```
  svg.append("path")
      .datum(stateMesh)
      .attr("fill", "none")
      .attr("stroke", "#777")
      .attr("stroke-width", 0.5)
      .attr("stroke-linejoin", "round")
      .attr("d", d3.geoPath(projection));
  // Append the hexagons (circles in this case).
  svg.append("g")
      .selectAll("circle")
      .data(bins)
      .join("circle")
      .attr("cx", d => d.x)
      .attr("cy", d => d.y)
      .attr("r", d => radius(d.meanAge))
      .attr("fill", d => color(d.length))
      .attr("stroke", d => d3.lab(color(d.length)).darker())
      .append("title")
      .text(d => `${d.meanAge.toFixed(2)} years mean age\n${d.length} stores`);
  return svg.node();
}
```

Explanation:

How the code works: Constructing the SVG container similarly as the 1st question.

Now here, I created the hexbin and bins with specific sizes and radius, as in this question the ages in a particular diameter is to be dealt with.

Calculated the mean age for each bin (group of stores) and sorted the bins by this mean age.

Sequential color scale for assigning the colors to the bins based on number of stores in each bin and square root scale for radius based on the mean age of stores in each bin.

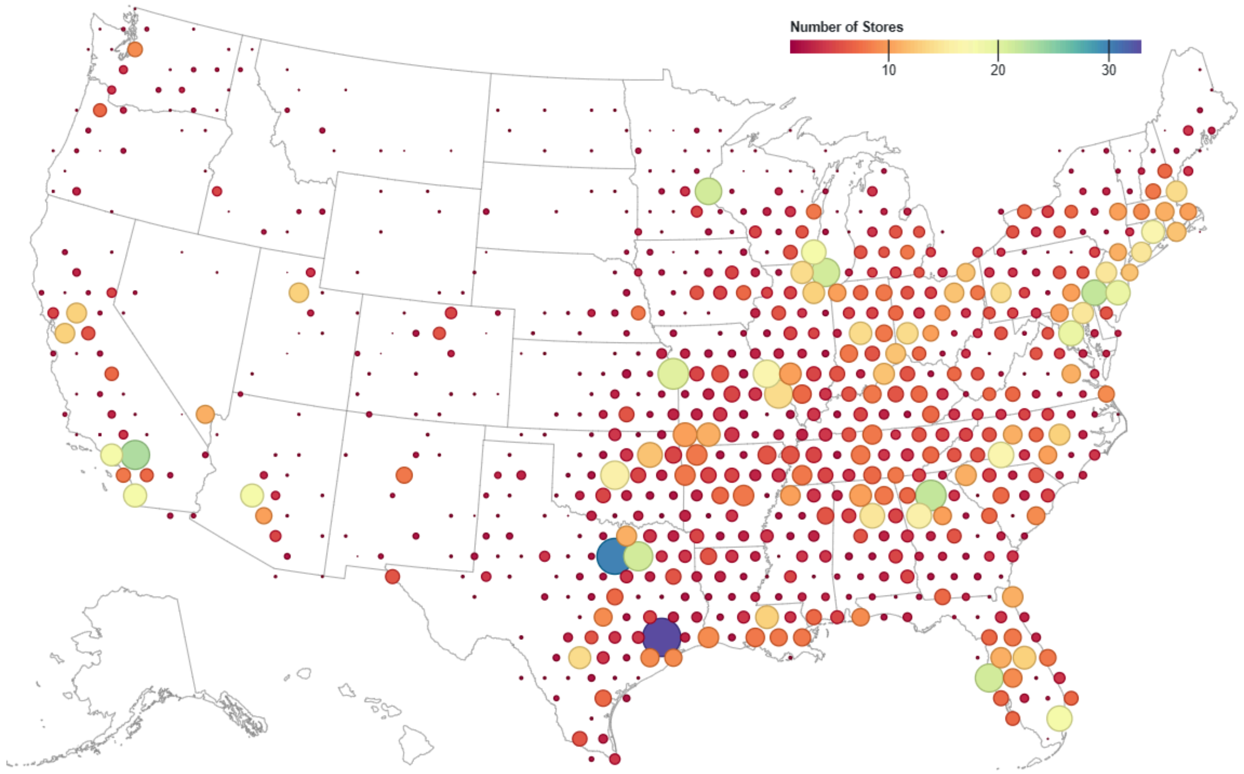
State mesh for the state boundaries.

Now circles were visualized over the “bins” data with coordinates of center as ‘cx’ and ‘cy’ and radius of circle as ‘r’. Color was filled based on the colorScale described above and a legend of this color assignment was shown at the top right corner.

The final visualization graph shows the walmart store locations with circles. The size of each circle corresponds to the mean age of stores of that hexbin, and the color intensity varies with the number of stores in each bin.

Question 3

Can you repeat the above work, but now you are aggregating the influenced age within a certain diameter and plotting the circles. Write a report on this. The ”influenced age” means that more number of older Walmart around, more is its weight-age as these are catered for the benefit of the rising population.



https://observablehq.com/d/2271aeadaba51824

Link to the observable notebook is above.

Code (d3.js library of javascript)

```
chart = {

  // Specify the map’s dimensions and projection.

  const width = 928;

  const height = 581;

  const projection = d3.geoAlbersUsa().scale(4 / 3 * width).translate([width / 2, height / 2]);

  // Create the container SVG.

  const svg = d3.create("svg")

      .attr("viewBox", [0, 0, width, height])

      .attr("width", width)

      .attr("height", height)

      .attr("style", "max-width: 100%; height: auto;");

  // Create the bins.

  const hexbin = d3.hexbin()

      .extent([[0, 0], [width, height]])

      .radius(10)

      .x(d => d.xy[0]) .y(d => d.xy[1]);

  const bins = hexbin(walmarts.map(d => ({

    xy: projection([d.longitude, d.latitude]),

    date: new Date(d.date) // Convert the date string to a Date object

  })))
```

```
.map(d => {
  // Calculate the median age for the bin
  const meanAge = d3.mean(d, d => (new Date()).getFullYear() - d.date.getFullYear());
  const influencedAge = meanAge*d.length; //total age
  return { length:d.length,x:d.x,y:d.y, meanAge:meanAge,influencedAge: influencedAge };
}).sort((a, b) => b.influencedAge - a.influencedAge);
 // Create the color scale.
const color = d3.scaleSequential(d3.extent(bins, d => d.length), d3.interpolateSpectral);
// Define the radius scale
const radius = d3.scaleSqrt()
    .domain( d3.extent(bins, d => d.influencedAge))
    .range([0, hexbin.radius() * Math.SQRT2]);
svg.append("g")
    .attr("transform", "translate(580,20)")
    .append(()=>legend({
      color,
      title: "Number of Stores",
      width: 260
      // Define appropriate tick values and format for the legend
    }));
// Append the state mesh.
```

```
  svg.append("path")
      .datum(stateMesh)
      .attr("fill", "none")
      .attr("stroke", "#777")
      .attr("stroke-width", 0.5)
      .attr("stroke-linejoin", "round")
      .attr("d", d3.geoPath(projection));
  // Append the hexagons (circles in this case).
  svg.append("g")
      .selectAll("circle")
      .data(bins)
      .join("circle")
      .attr("cx", d => d.x)
      .attr("cy", d => d.y)
      .attr("r", d => radius(d.influencedAge))
      .attr("fill", d => color(d.length))
      .attr("stroke", d => d3.lab(color(d.length)).darker())
      .append("title")
      .text(d => `${d.influencedAge.toFixed(2)} years mean age\n${d.length} stores`);
  return svg.node();
}
```

Explanation :

How code works: Here also we are creating the SVG container and the hexbins and bins. Here in the bins data, an attribute of influencedAge is also added. This is calculated as the total sum of ages of stores in that particular bin which I got by multiplying the mean age of the bin by the number of stores in that bin. Sort the bins by the 'influencedAge' in descending order.

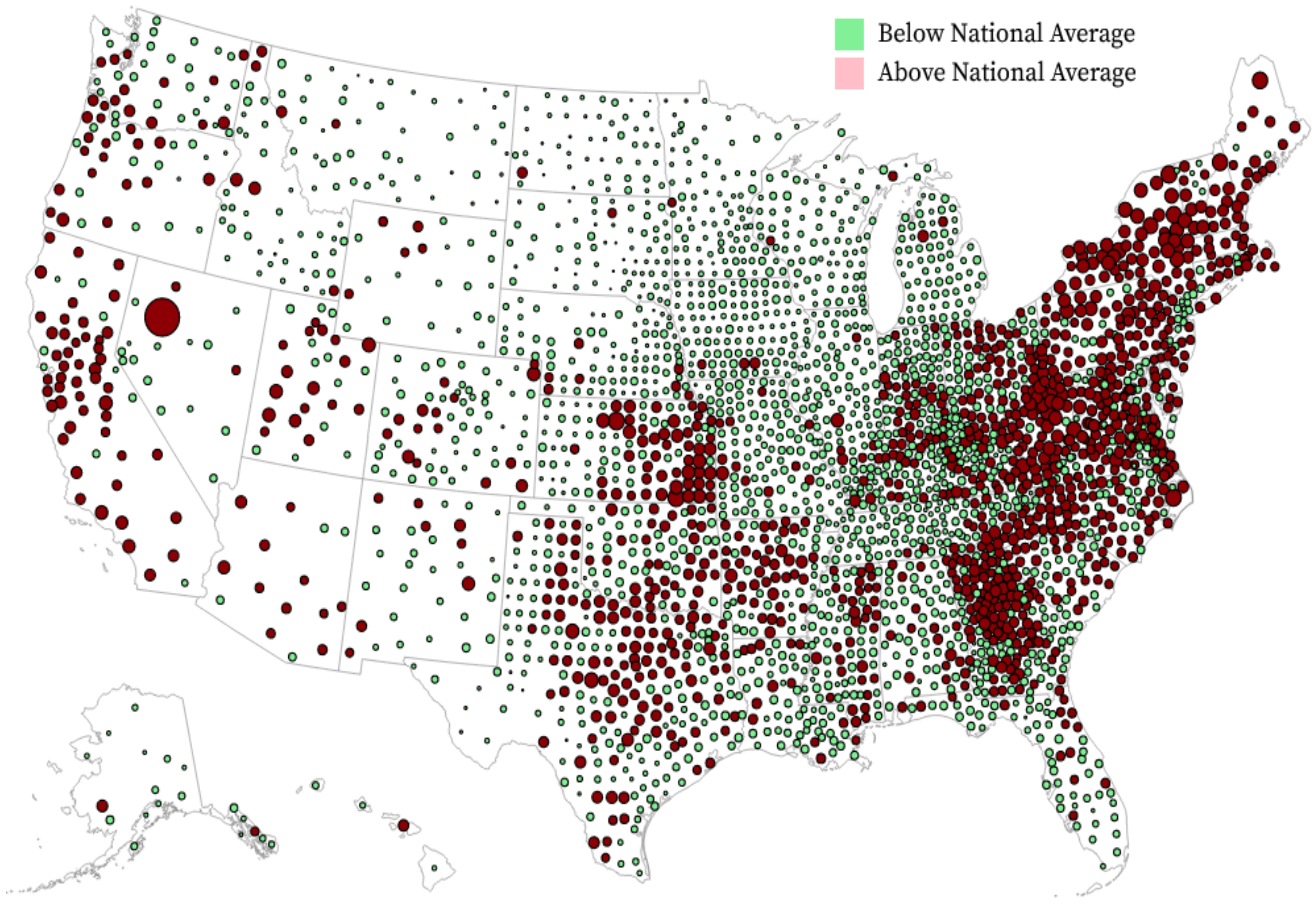
The color is according to the number of stores in the bin and the radius is according to the influencedAge of each bin.

A legend is provided to show the metric for color assignment.

Statemesh is drawn and circles are also drawn using the color and radius scales described above. SVG element is returned.

Question 4

Now consider the dataset ”Coronavirus (Covid-19) Data of United States (USA)”. Your task is to identify the locations which are constantly under threat (rising number of cases and greater than national average) for a time window. Plot this on the JavaScript library.



https://observablehq.com/d/0af78a412bbc1429

Link to the observable notebook is above.

Code (d3.js library of javascript)

```
covid =
d3.csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv")

data = await
d3.csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv");

parsedData = data.map( d => (

 {

...d,

date: d.date,

county: d.county,

code: d.fips,

state: d.state,

cases: +d.cases,

deaths: +d.deaths

 }

));

startDate = new Date("2020-12-25");

endDate = new Date("2021-01-01");

startDateData = filteredData.filter(d => new Date(d.date).getTime() === startDate.getTime());

aggData1 = d3.rollup(startDateData, v => ({
  totalCases: d3.sum(v, d => d.cases),

  totalDeaths: d3.sum(v, d => d.deaths),

  deathPercentage: (d3.sum(v, d => d.deaths) / d3.sum(v, d => d.cases)) * 100

}));

endDateData = filteredData.filter(d => new Date(d.date).getTime() === endDate.getTime());

aggData2 = d3.rollup(endDateData, v => ({

  totalCases: d3.sum(v, d => d.cases),

  totalDeaths: d3.sum(v, d => d.deaths),

  deathPercentage: (d3.sum(v, d => d.deaths) / d3.sum(v, d => d.cases)) * 100

}));

nationalaverage_case = (aggData2.totalCases-aggData1.totalCases)*100/(aggData1.totalCases)

nationalaverage_death=
(aggData2.totalDeaths-aggData1.totalDeaths)*100/(aggData1.totalDeaths)

g1Data = d3.rollups(startDateData,

  (group) => ({

totalCases: d3.sum(group, d => d.cases),

totalDeaths: d3.sum(group, d => d.deaths),

deathPercentage: (d3.sum(group, d => d.deaths)/d3.sum(group, d => d.cases))*100

  }),

  d => d.county, d => d.fips // Group by county and fips

);
```

```
g2Data = d3.rollups(endDateData,
  (group) => ({
totalCases: d3.sum(group, d => d.cases),
totalDeaths: d3.sum(group, d => d.deaths),
deathPercentage: (d3.sum(group, d => d.deaths)/d3.sum(group, d => d.cases))*100
  }),
  d => d.county, d => d.fips // Group by county and fips
);
gData1 = d3.rollup(startDateData, v => ({
  totalCases: d3.sum(v, d => d.cases),
  totalDeaths: d3.sum(v, d => d.deaths),
  deathPercentage: (d3.sum(v, d => d.deaths) / d3.sum(v, d => d.cases)) * 100
}), d => d.fips);
gData2 = d3.rollup(endDateData, v => ({
  totalCases: d3.sum(v, d => d.cases),
  totalDeaths: d3.sum(v, d => d.deaths),
  deathPercentage: (d3.sum(v, d => d.deaths) / d3.sum(v, d => d.cases)) * 100
}), d => d.fips);
gData3 = new Map();
// Iterate over the keys of gData2
gData2.forEach((value, key) => {
  const oldCases = gData1.has(key) ? gData1.get(key).totalCases : 0; // Get the old cases or default to 0

  const newCases = value.totalCases;

  const oldDeaths = gData1.has(key) ? gData1.get(key).totalDeaths : 0; // Get the old cases or default to 0

  const newDeaths = value.totalDeaths;

  // Compute the percentage increase

  const increase = oldCases ? ((newCases - oldCases) / oldCases) * 100 : 0;

  const inc = oldDeaths ? ((newDeaths - oldDeaths) / oldDeaths) * 100 : 0;

  // Set the increase in the new map

  gData3.set(key, {

totalCases: newCases,

totalDeaths: value.totalDeaths,

deathPercentage: value.deathPercentage,

caseIncreasePercentage: increase,

deathIncreasePercentage: inc

  });

});

lat_log = d3.csv("https://gist.githubusercontent.com/russellsamora/12be4f9f574e92413ea3f92ce1bc58e6/raw/3f18230058afd7431a5d394dab7eeb0aafd29d81/us_county_latlng.csv");

chart = {

  const svg = d3.create("svg")
```

```
        .attr("viewBox", [0, 0, 960, 600]);
 svg.append("path")
        .datum(topojson.merge(us, us.objects.lower48.geometries))
        .attr("fill", "#ddd")
        .attr("d", d3.geoPath());
 svg.append("path")
        .datum(topojson.mesh(us, us.objects.lower48, (a, b) => a !== b))
        .attr("fill", "none")
        .attr("stroke", "white")
        .attr("stroke-linejoin", "round")
        .attr("d", d3.geoPath());
 const g = svg.append("g")
        .attr("fill", "none")
        .attr("stroke", "black");
 const dot = g.selectAll("circle")
.data(data)
.join("circle")
        .attr("transform", d => `translate(${d})`);
 svg.append("circle")
        .attr("fill", "blue")
        .attr("transform", `translate(${data[0]})`)
```

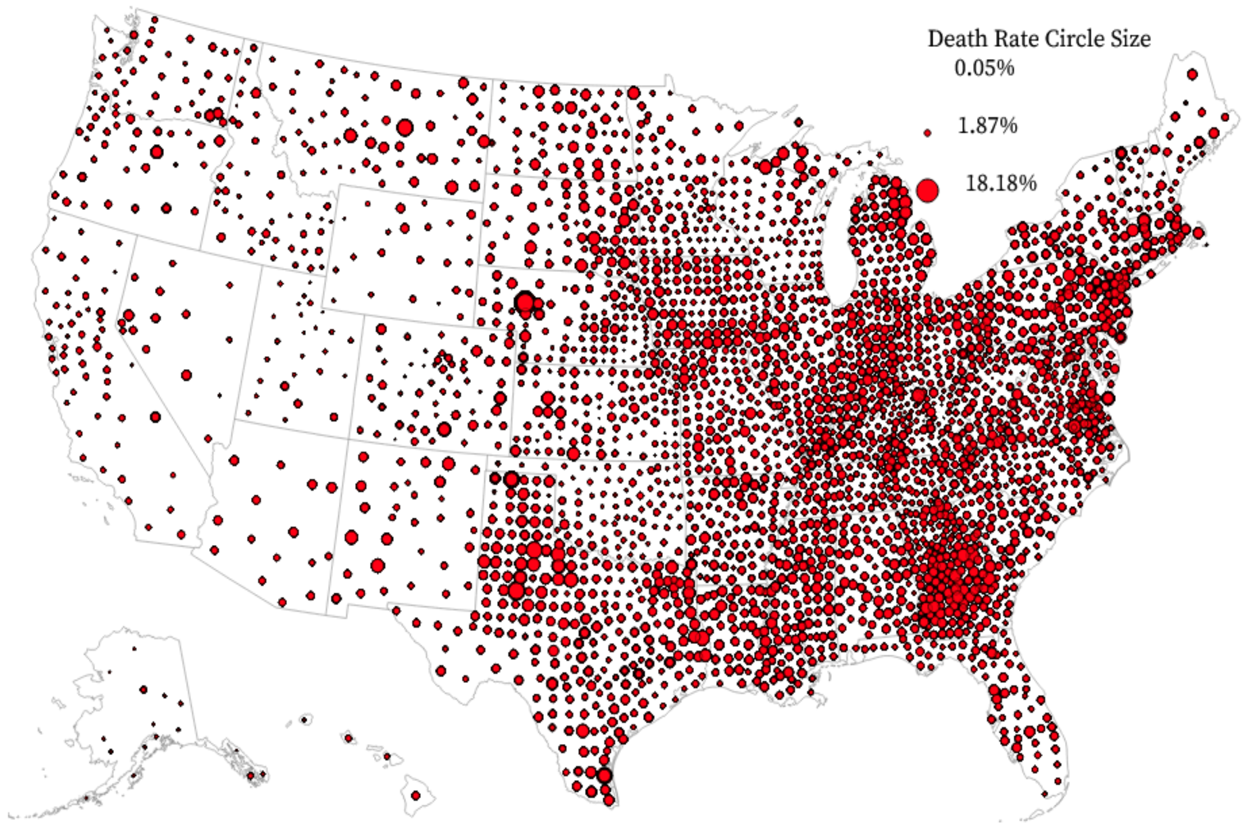
```
            .attr("r", 3);


  let previousDate = -Infinity;

  return Object.assign(svg.node(), {

update(date) {

            dot // enter

            .filter(d => d.date > previousDate && d.date <= date)

            .transition().attr("r", 3);

            dot // exit

            .filter(d => d.date <= previousDate && d.date > date)

            .transition().attr("r", 0);

            previousDate = date;

}

  });

}

update = chart.update(date)

data = (await FileAttachment("walmart.tsv").tsv())

  .map(d => {

const p = projection(d);

p.date = parseDate(d.date);

return p;
```

```
  })
  .sort((a, b) => a.date - b.date)
parseDate = d3.utcParse("%m/%d/%Y")
projection = d3.geoAlbersUsa().scale(1280).translate([480, 300])
geo_data = await
d3.csv("https://gist.githubusercontent.com/russellsamora/12be4f9f574e92413ea3f92ce1bc58e6/raw/3f18230058afd7431a5d394dab7eeb0aafd29d81/us_county_latlng.csv");
geoMapping = geo_data.reduce((acc, cur) => {
  acc[cur.fips_code] = { lat: +cur.lat, lon: +cur.lng, name: cur.name };
  return acc;
}, {});
mergedData = [...gData3].map(([county, dataArray]) => {
  const fipsCode = county;
  const data = dataArray;
  const geoInfo = geoMapping[fipsCode] || {};
  return {
...data,
fips_code: fipsCode,
name: geoInfo.name,
latitude: geoInfo.lat,
longitude: geoInfo.lon
  };
```

```
});

chart1 = {

  const width = 928;

  const height = 581;

  const projection = d3.geoAlbersUsa().scale(4 / 3 * width).translate([width / 2, height / 2]);

  const svg = d3.create("svg")

            .attr("viewBox", [0, 0, width, height])

            .attr("width", width)

            .attr("height", height)

            .attr("style", "max-width: 100%; height: auto;");

  const color = d3.scaleThreshold()

.domain([nationalaverage_case])

.range(["lightgreen", "darkred"])

 const legendLabels = ['Below National Average', 'Above National Average'];

const legendColorScale = d3.scaleOrdinal()

  .domain(legendLabels)

  .range(['lightgreen', 'pink']); // green for below, red for above

const legend = svg.append('g')

  .attr("transform", "translate(580,20)");

legend.selectAll(null)

  .data(legendLabels)
  .enter()

  .append('rect')

  .attr('y', (d, i) => i * 25)

  .attr('width', 20)

  .attr('height', 20)

  .attr('fill', d => legendColorScale(d));

legend.selectAll(null)

  .data(legendLabels)

  .enter()

  .append('text')

  .attr('x', 30) // slightly more space for the text

  .attr('y', (d, i) => i * 25 + 15)

  .text(d => d);

  const IncScale = d3.scaleSqrt()

.domain([0, d3.max(mergedData, d => d.caseIncreasePercentage)])

.range([0, 12]);

  svg.append("path")

            .datum(stateMesh)

            .attr("fill", "none")

            .attr("stroke", "#777")

            .attr("stroke-width", 0.5)
            .attr("stroke-linejoin", "round")
            .attr("d", d3.geoPath(projection));
  svg.selectAll('circle')
.data(mergedData)
.enter()
.append('circle')
            .attr('cx', d => {
            const coords = projection([d.longitude, d.latitude]);
            return coords ? coords[0] : null;
            })
            .attr('cy', d => {
            const coords = projection([d.longitude, d.latitude]);
            return coords ? coords[1] : null;
            })
            .attr('r', d => IncScale(d.caseIncreasePercentage))
            .attr('fill', d => color(d.caseIncreasePercentage))
            .attr('stroke', 'black')
.append("title")
            .text(d => `${d.caseIncreasePercentage.toLocaleString()}% increase in last 7 days`);
  return svg.node();
}
```

Question 5

Now consider the dataset ”Coronavirus (Covid-19) Data of United States (USA)”. Your task is to identify the regions where the plots are proportional to the percentage of death compared to affected people for a time window. Plot this on the JavaScript library.



https://observablehq.com/d/2136d4d7523b4986

Link to the observable notebook.

Code (d3.js library of javascript)

```
chart = {
  // Specify the map’s dimensions and projection.
  const width = 928;
  const height = 581;
  const projection = d3.geoAlbersUsa().scale(4 / 3 * width).translate([width / 2, height / 2]);
  // Create the container SVG.
  const svg = d3.create("svg")
            .attr("viewBox", [0, 0, width, height])
            .attr("width", width)
            .attr("height", height)
            .attr("style", "max-width: 100%; height: auto;");
//mapped data in a pre defined time scale
const filteredData = mapped
  .filter(d => d.longitude != null && d.latitude != null) // Check for non-null coordinates
  .filter(d=>d.cases> 0 && d.deaths/d.cases>0) // Exclude entries without cases or deaths
  .map(d => ({
...d,
projectedCoord: projection([d.longitude, d.latitude]) // Project the coordinates
  }))
```

```
 .filter(d => d.projectedCoord != null); // Exclude entries that can't be projected
 const deathRatio=filteredData.map(d=>({
...d,
deathRate: (d.deaths/d.cases)*100 //percentage
 }));
 Const radiusScale=d3.scaleSqrt().domain(d3.extent(deathRatio,d=>d.deathRate)).range([0,8]);
 // Append the state mesh.
 // for the state borders
 svg.append("path")
        .datum(stateMesh)
        .attr("fill", "none")
        .attr("stroke", "#777")
        .attr("stroke-width", 0.5)
        .attr("stroke-linejoin", "round")
        .attr("d", d3.geoPath(projection));
 // Create and append the circles to the SVG
svg.selectAll("circle")
 .data(deathRatio)
 .join("circle")
.attr("cx", d => projection([d.longitude, d.latitude])[0])
.attr("cy", d => projection([d.longitude, d.latitude])[1])
```

```
.attr("r", d => radiusScale(d.deathRate)) // Size circles based on death rate

.attr("fill", "red") // Choose your color

.attr("opacity", 0.6) // Optional: make the circles semi-transparent

.attr("stroke","black")

.append("title")

.text(d => ${d.region}: ${d.deaths} deaths, ${d.cases} cases, Death rate:
${d.deathRate.toFixed(2)}%);

const legendMargin = { right:20, top: 150 };

  const legend = svg.append("g")

  .attr("transform", translate(${width - legendMargin.right}, ${height - legendMargin.top}));

const exampleDeathRates = [d3.min(deathRatio, d => d.deathRate),

                           d3.mean(deathRatio, d => d.deathRate),

                           d3.max(deathRatio, d => d.deathRate)];

// Create a group for the size legend

const sizeLegend = svg.append("g")

  .attr("transform", "translate(680, 60)");  // Adjust position to fit your chart

// Add legend title for size

sizeLegend.append("text")

  .attr("class", "legend-size-title")

  .attr("x", 0)

  .attr("y", -20)
  .style("text-anchor", "left")

  .text("Death Rate Circle Size");

// Append circles for each example size

exampleDeathRates.forEach((rate, i) => {

  const yPosition = i * 40;  // Adjust vertical spacing to fit your chart

  sizeLegend.append("circle")

.attr("cx", 0)

.attr("cy", yPosition)

.attr("r", radiusScale(rate))

.attr("fill", "red")

.attr("stroke", "black");

  sizeLegend.append("text")

.attr("x", 20 + radiusScale(rate))

.attr("y", yPosition)

.style("alignment-baseline", "middle")

.text(${rate.toFixed(2)}%);

});

  return svg.node();

}
```

Creating modified data by filtering according to a time stamp

```
mapped={
```

```
const start = new Date('2021-01-01');

const end = new Date('2021-03-05');


// Assuming 'covid' is an array of COVID data objects

// and 'countyy' is an array of county data objects with 'lat' and 'long' properties.

const countymap = new Map(countyy.map(d => [d.fips, d]));


// Filter COVID data by date

const covidFiltered = covid.filter(d => {

  const date = new Date(d.date);

  return date >= start && date <= end;

});


// Enrich the filtered COVID data with latitude and longitude from county data

const covidData = covidFiltered.map(d => {

  const countyData = countymap.get(d.fips);

  if (countyData) {

return {

            ...d,

            latitude: countyData.lat,

            longitude: countyData.long,
};
  }
  return d;
});
  return covidData;
}
```

Loading the data files

```
covid= {
  const parseDate=d3.utcParse("%Y-%m-%d");
  return
d3.csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv")
  .then(data=>data.map(d=>({
county:d.county,
state:d.state,
cases:d.cases,
fips:d.fips,
deaths:d.deaths,
date:parseDate(d.date)
  })));
}
countyy={
  return
d3.csv("https://gist.githubusercontent.com/russellsamora/12be4f9f574e92413ea3f92ce1bc58e6/raw/3f18230058afd7431a5d394dab7eeb0aafd29d81/us_county_latlng.csv")
  .then(data=>data.map(d=>({
county:d.name,
fips:d.fips_code,
long:d.lng,
lat:d.lat
  })))
}
```