Student Declaration of Authorship

Course code and name:	**F20DV – DATA VISUALIZATION AND ANALYTICS**
Type of assessment:	**Individual**
Coursework Title:	**Lab 4: Dataset Visualisation & Analytics**
Student Name:	**Ayushi Madhukumar Amin**
Student ID Number:	**H00331154**

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):* *Ayushi Madhukumar Amin*

Date: *04/04/2022*

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.

Lab 4

F20DV – DATA VISUALIZATION AND ANALYTICS

CW - 4

Name: Ayushi MadhuKumar Amin
Registration Number: H00331154
Program: Bsc. Computer Science (Year – 4)
Campus: Heriot-Watt University Dubai
Date: April 4, 2022
Code and results demonstrated to: Dr. Ryad Soobhany

Introduction:

The image above is a dashboard designed to give users a detailed look at crime figures in the United States during the last 55 years (1965-2019).

Users are offered two options to choose from, i.e., the two major crime kinds – **Property and Violent**, and the users can select the one they want to investigate further. Each of the two major crime categories comprises of subcategories, which are displayed in the table below.

Property	**Violent**
Burglary	Assault
Larceny	Murder
Motor	Rape
-	Robbery

On choosing an option, users are provided with multiple graphs that show the correlation between the average population and the subcategories (eg: Burglary, Murder, etc.) for a particular year. In addition, the dashboard displays a ranking of states based on how high the rate of the chosen crime (Property or Violent) is in that state. A donut chart depicting the top crime subcategories for a particular year is also shown.

The dashboard is composed of the following layouts:

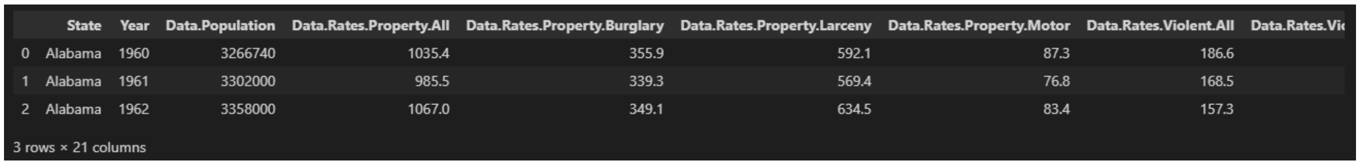
1. Choropleth Map of the United States
2. Donut chart – Shows the top crime subcategories for a particular year
3. Circular Bar chart – Displays the ranking of the states, ordered by the rate of the chosen crime category (Property or Violent)
4. Scatter plot – Shows the correlation between a crime subcategory and the overall population
5. Area Chart – Displays the evolution of crime subcategories over the course of 55 years for a selected country

Data:

For this dashboard, the main data has been taken from the following page (*The CORGIS Dataset Project*): https://corgis-edu.github.io/corgis/csv/state_crime/

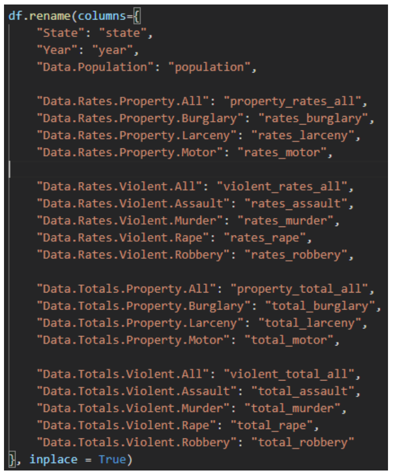
The dataset is open source (Bart et al., 2017) and contains data on crime rates and totals for all 50 states and the federal district - Washington D.C. (District of Columbia). The dataset ranges from 1960 to 2019 and its dimensions are – 3116 rows and 21 columns.

The dataset was processed with Python and Pandas before being used for the dashboard, and the image below demonstrates how the original dataset looks when loaded into a DataFrame.



```
     State    Year  Data.Population  Data.Rates.Property.All  Data.Rates.Property.Burglary  Data.Rates.Property.Larceny  Data.Rates.Property.Motor  Data.Rates.Violent.All  Data.Rates.Vi
0  Alabama    1960          3266740                   1035.4                         355.9                        592.1                       87.3                   186.6
1  Alabama    1961          3302000                    985.5                         339.3                        569.4                       76.8                   168.5
2  Alabama    1962          3358000                   1067.0                         349.1                        634.5                       83.4                   157.3

3 rows × 21 columns
```

Because certain datasets contain duplicate data, the Python program uses the duplicated() Pandas function to see if there are any duplicates in the DataFrame. Due to the truncation of the result, a for loop was constructed to print which rows contained duplicate data. The for loop did not return any rows, hence there were no duplicate values present in the dataset.



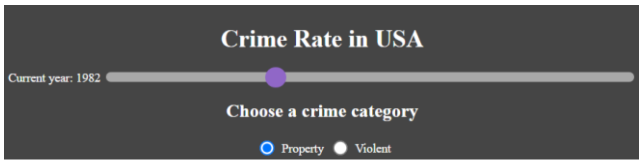
```
df.rename(columns={
    "State": "state",
    "Year": "year",
    "Data.Population": "population",

    "Data.Rates.Property.All": "property_rates_all",
    "Data.Rates.Property.Burglary": "rates_burglary",
    "Data.Rates.Property.Larceny": "rates_larceny",
    "Data.Rates.Property.Motor": "rates_motor",

    "Data.Rates.Violent.All": "violent_rates_all",
    "Data.Rates.Violent.Assault": "rates_assault",
    "Data.Rates.Violent.Murder": "rates_murder",
    "Data.Rates.Violent.Rape": "rates_rape",
    "Data.Rates.Violent.Robbery": "rates_robbery",

    "Data.Totals.Property.All": "property_total_all",
    "Data.Totals.Property.Burglary": "total_burglary",
    "Data.Totals.Property.Larceny": "total_larceny",
    "Data.Totals.Property.Motor": "total_motor",

    "Data.Totals.Violent.All": "violent_total_all",
    "Data.Totals.Violent.Assault": "total_assault",
    "Data.Totals.Violent.Murder": "total_murder",
    "Data.Totals.Violent.Rape": "total_rape",
    "Data.Totals.Violent.Robbery": "total_robbery"
}, inplace = True)
```

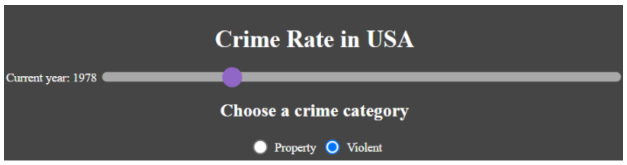
To increase readability and for easier access in the JavaScript programs, the column names of the DataFrame were modified using Pandas rename() function. The code snippet on the right demonstrates how and to what the columns were renamed.

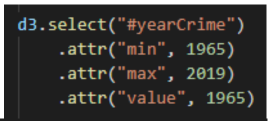
Aside from 50 states and the federal district (Washington, D.C.), the dataset featured a few rows that displayed the overall data collected across the country. Since the dashboard only deals with data collected per state and district, the rows having the column "state" value as 'United States" were dropped. Additionally, the state "New York" only has data from 1965 to 2019, hence the Python program only extracts the data collected from 1965 onwards. The resulting subset was stored in a DataFrame, which was then converted to a CSV file. This CSV file served as the primary dataset for the dashboard.

The finance dataset used for processed in the exact same way and it has been taken from the following page: https://corgis-edu.github.io/corgis/csv/finance/

Dashboard Header:



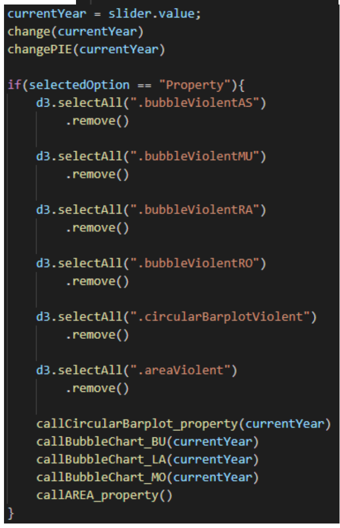




```
d3.select("#yearCrime")
    .attr("min", 1965)
    .attr("max", 2019)
    .attr("value", 1965)
```

The dashboard uses HTML's range slider to present the crime rate scene in the United States from 1965 to 2019. The slider's minimum and maximum values are 1965 and 2019, respectively, and they're set using D3 and the ID allocated to the <input> tag.

By sliding the purple marker back and forth over the HTML slider, users can see crime rate statistics for a specified year. The selected year will be displayed on the left side of the slider and will be saved in a local variable that will be used as an argument to multiple local and global functions present in various JavaScript files. These functions are used for generating the multiple layouts that appear on the dashboard.

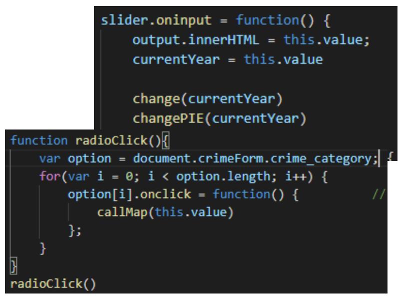


```
currentYear = slider.value;
change(currentYear)
changePIE(currentYear)

if(selectedOption == "Property"){
    d3.selectAll(".bubbleViolentAS")
        .remove()

    d3.selectAll(".bubbleViolentMU")
        .remove()

    d3.selectAll(".bubbleViolentRA")
        .remove()

    d3.selectAll(".bubbleViolentRO")
        .remove()

    d3.selectAll(".circularBarplotViolent")
        .remove()

    d3.selectAll(".areaViolent")
        .remove()

    callCircularBarplot_property(currentYear)
    callBubbleChart_BU(currentYear)
    callBubbleChart_LA(currentYear)
    callBubbleChart_MO(currentYear)
    callAREA_property()
}
```

After initially picking a crime category, the slider by default selects the year 1965, and the dashboard displays crime statistics for that year. As discussed above, 1965 is stored in a local variable which is sent as an argument to multiple functions and the layouts are generated accordingly. The code snippet on the right shows an example of how the selected year (taken from the slider) is stored in a local variable named "currentYear" and how it serves as an argument for various functions.

Additionally, the slider uses the “oninput” attribute to listen for input events, which are subsequently stored in the local variable “currentYear”. This is done to modify the dashboard layouts so that the statistics are displayed precisely for the selected year.

Aside from the slider, the user can select a crime type by pressing one of the two radio buttons - Property or Violent. The “onclick” attribute is used by the local JavaScript function “radioClick()” to listen for click events. The “radioClick()” function sends the selected option to a global function whenever the user picks a radio button. This function was created to generate specific layouts for the selected crime category.



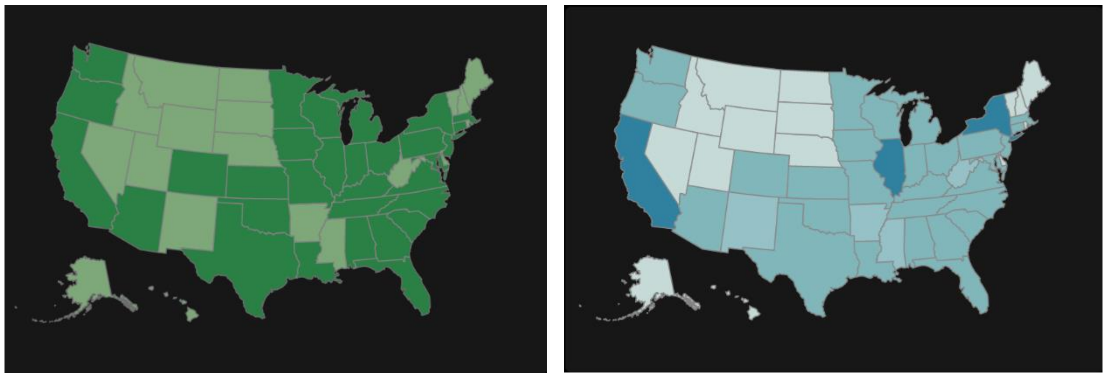
```
slider.oninput = function() {
    output.innerHTML = this.value;
    currentYear = this.value

    change(currentYear)
    changePIE(currentYear)
```

```
function radioClick(){
    var option = document.crimeForm.crime_category;
    for(var i = 0; i < option.length; i++) {
        option[i].onclick = function() {          //
            callMap(this.value)
        };
    }
}
radioClick()
```

Layouts:

<u>Layout 1</u>: *Choropleth Map of the United States*

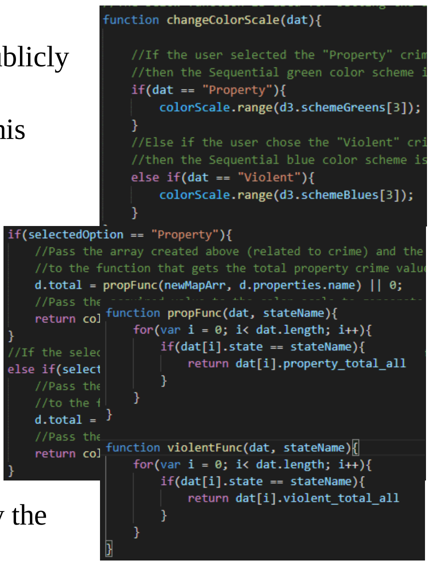
(a) (b)



The above two images illustrate how the choropleth map appears (for the year 1965) when the (a) Property crime category is selected and (b) Violent crime category is chosen.

For generating the basic map structure, the program makes use of a geoJSON file that is publicly available on the following GitHub repository: https://github.com/PublicaMundi/MappingAPI/blob/master/data/geojson/us-states.json . This JSON file contains coordinates of each US state along with their respective names.

For building the projection of the map, the program exploits D3’s geoAlbersUsa() projection. Initially, the geoMercator() projection was used for the map. However, due to the distance between the main body of the US and Alaska and Hawaii, the geoMercator() projection was not ideal as a large amount of space on the SVG object was wasted and the map appeared diminished, even after adjusting the translate(). The geoAlbersUsa() projection positions the map of Alaska and Hawaii below the main body of the US. This ensures a large amount of space isn’t wasted on the SVG object. For coloring the map, the defined color scale makes use of D3’s scaleThreshold(), where the domain ranges from 1000 to 32000. The range was dynamically set based on what crime category was chosen by the



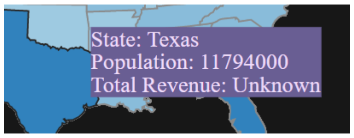
```
function changeColorScale(dat){

    //If the user selected the "Property" crim
    //then the Sequential green color scheme i
    if(dat == "Property"){
        colorScale.range(d3.schemeGreens[3]);
    }
    //Else if the user chose the "Violent" cri
    //then the Sequential blue color scheme is
    else if(dat == "Violent"){
        colorScale.range(d3.schemeBlues[3]);
    }
```

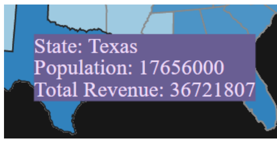
```
if(selectedOption == "Property"){
    //Pass the array created above (related to crime) and the
    //to the function that gets the total property crime value
    d.total = propFunc(newMapArr, d.properties.name) || 0;
    //Pass the
    return col
}
//If the selec
else if(select
    //Pass the
    //to the f
    d.total =
    //Pass the
    return col
}
```

```
function propFunc(dat, stateName){
    for(var i = 0; i< dat.length; i++){
        if(dat[i].state == stateName){
            return dat[i].property_total_all
        }
    }
}
```

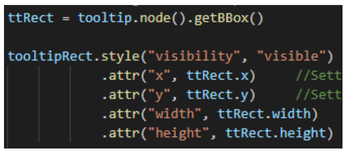
```
function violentFunc(dat, stateName){
    for(var i = 0; i< dat.length; i++){
        if(dat[i].state == stateName){
            return dat[i].violent_total_all
        }
    }
}
```

user. As seen from the above two images of the US map, the choropleth map representing the “Property” and “Violent” crime category appear in shades of green and blue, respectively. If the user selects the Property crime category, D3’s sequential green color scheme, d3.schemeGreen() is serves as the range for the color scale. The d3.schemeBlues() sequential blue color scheme is set as the range of the color scale, provided the user chooses the Violent crime category. Each state was colored accordingly using the “fill” attribute and an Object array. Values recorded on the selected year (taken from the slider) were extracted using the “filter()” function, that generated a new object array. This object array is passed to the function used for getting the total “Property” or “Violent” value (based on which option was selected) for a particular state. The result is passed to the color scale (created for coloring the choropleth map) and it colors the states accordingly.

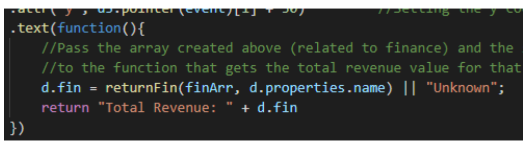




The image on the right displays what appears when the mouse hovers over any state. A tooltip was defined for both the text and the rectangle that appears behind the text element. The code makes use of the getBBox() function so that the rectangle’s width and height is set according to the dimensions of the SVG text element and it’s coordinates as well are set as per the x and y coordinates of the SVG text element. The logic used for dynamically getting the “Population” and “Total Revenue” value of a state, is similar to the one used for getting the total Property or Violent value for the color scale. For example, in the case of “Total Revenue”, an object array called “finArr” is created by extracting out the records taken on a particular year (taken from the slider). A local function called “returnFin()” is implemented using the similar logic as the function defined for obtaining the total “Property” or “Violent” value for the color scale. This function, instead, returns a state’s total revenue for a given year. If there is no record of a state’s total revenue in a given year, the value “Unknown” is returned instead.

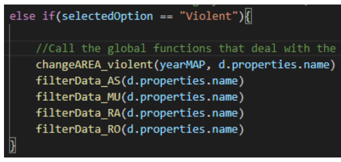


```
ttRect = tooltip.node().getBBox()

tooltipRect.style("visibility", "visible")
            .attr("x", ttRect.x)      //Sett
            .attr("y", ttRect.y)      //Sett
            .attr("width", ttRect.width)
            .attr("height", ttRect.height)
```



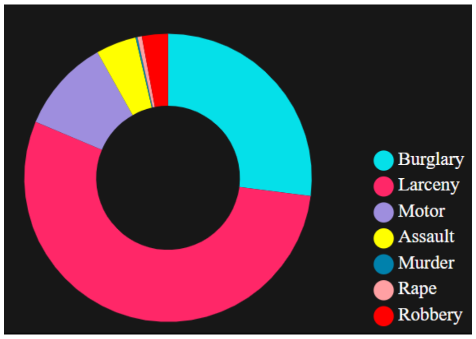
```
.text(function(){
    //Pass the array created above (related to finance) and the
    //to the function that gets the total revenue value for that
    d.fin = returnFin(finArr, d.properties.name) || "Unknown";
    return "Total Revenue: " + d.fin
})
```

In addition, the code listens for click events. A local function called “onMouseClick()” is triggered if any of the states is clicked. This function checks whether the selected crime category is “Property” or “Violent”, and then calls and executes relevant global functions based on the selected crime type.

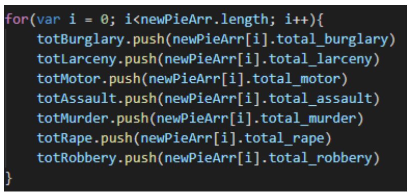


```
else if(selectedOption == "Violent"){

    //Call the global functions that deal with the
    changeAREA_violent(yearMAP, d.properties.name)
    filterData_AS(d.properties.name)
    filterData_MU(d.properties.name)
    filterData_RA(d.properties.name)
    filterData_RO(d.properties.name)
}
```

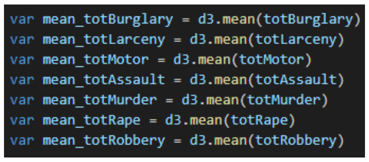
<u>Layout 2</u>: *Donut Chart – Shows the top crime subcategories for a particular year*



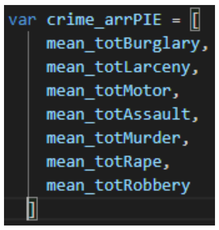
The dashboard uses a donut chart to show the top crime subcategories in the United States for a given year. The proportion of the average total number of crimes committed for each type of crime subcategory (Burglary, Murder, Assault, etc.) was displayed using a donut chart. This layout gives the user a better notion of which crime subcategory has the most crime records for a given year.



```
for(var i = 0; i<newPieArr.length; i++){
    totBurglary.push(newPieArr[i].total_burglary)
    totLarceny.push(newPieArr[i].total_larceny)
    totMotor.push(newPieArr[i].total_motor)
    totAssault.push(newPieArr[i].total_assault)
    totMurder.push(newPieArr[i].total_murder)
    totRape.push(newPieArr[i].total_rape)
    totRobbery.push(newPieArr[i].total_robbery)
}
```



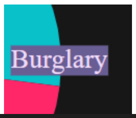
```
var mean_totBurglary = d3.mean(totBurglary)
var mean_totLarceny = d3.mean(totLarceny)
var mean_totMotor = d3.mean(totMotor)
var mean_totAssault = d3.mean(totAssault)
var mean_totMurder = d3.mean(totMurder)
var mean_totRape = d3.mean(totRape)
var mean_totRobbery = d3.mean(totRobbery)
```



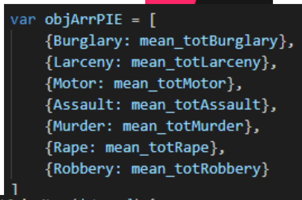
```
var crime_arrPIE = [
    mean_totBurglary,
    mean_totLarceny,
    mean_totMotor,
    mean_totAssault,
    mean_totMurder,
    mean_totRape,
    mean_totRobbery
]
```

A global function called “changePie()” (defined in the donut chart layout JavaScript file) is triggered when the user selects a year by using the slider present on the header of the dashboard. Using the “filter()” function, an object array called “newPieArr” is created by extracting only the records taken on the chosen year. As each proportion in a donut chart represents the average total number of records collected for a crime subcategory in a given year, the code stores the total value for each subcategory in seven different arrays. The average of each of these seven arrays was computed separately using D3’s mean() function, and the results were saved in an array called “crime_arrPIE”. This array was used for generating the donut chart. The code snippets above show how this task was achieved.

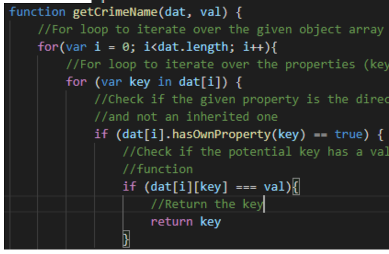
When the user selects one of the crime categories (Property or Violent) or changes the current year (using the slider), the donut chart animates (spins) in anticlockwise direction and updates/appears on the dashboard. This is done by using the transition() and duration() functions, where the duration is set to 1000 ms. Additionally, for making the chart segments to spin and appear, the interpolate() function and transition’s attrTween() was used. Within the attrTween(), a custom interpolator was created using the start and end angles of the donut chart. Using these functions, helped achieve the smooth transition of the donut chart. An ordinal scale of seven colors was defined for coloring each wedge.



The image on the right displays what appears when the mouse hovers over any wedge (the wedge becomes opaque). Like in the Choropleth map implementation above, the donut chart makes use of two tooltips that are defined for an SVG text and rect element. In addition to this, the BBox() function is used for setting the dimensions and coordinates of the SVG rectangle as per the text element. The following logic was used for getting the name of the resepcetive crime subcategory, when hovering over a certain wedge. After computing the average of each subcategory, an object array called objArrPIE was initialized. Within this object array, the name of the subcategory served as the “key” and its value was set to be the calculated average value. A screenshot of the object array is shown on the right. A local function called “getCrimeName()” was created to get the key of a given value. The code first checks if the specified property (the potential “key” object) is the direct property of the object . If this is true, the code then checks whether the property has a value equal to the one provided as an argument to the “getCrimeName()” function. When hovering over a wedge, its value and the “objArrPIE” is sent to “getCrimeName()” function to get the key, that appears when the mouse hovers over the wedge.



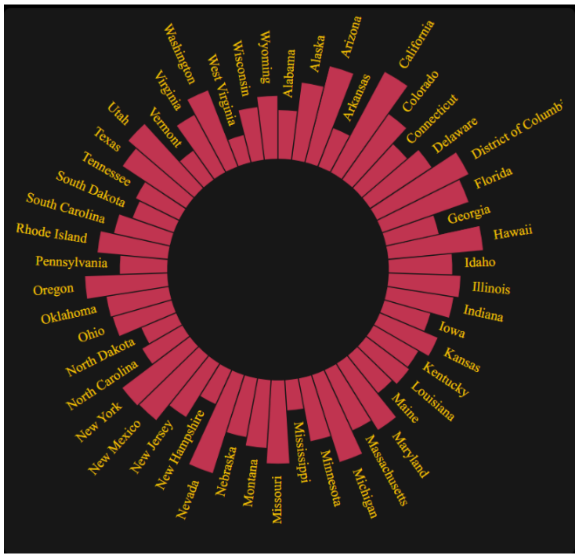
```
var objArrPIE = [
    {Burglary: mean_totBurglary},
    {Larceny: mean_totLarceny},
    {Motor: mean_totMotor},
    {Assault: mean_totAssault},
    {Murder: mean_totMurder},
    {Rape: mean_totRape},
    {Robbery: mean_totRobbery}
]
```



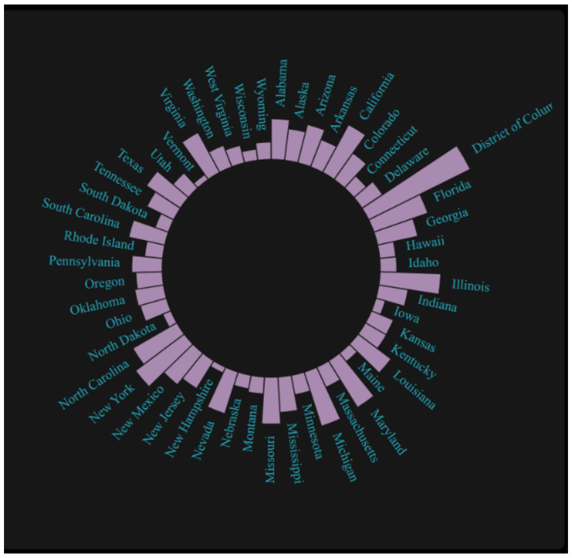
```
function getCrimeName(dat, val) {
    //For loop to iterate over the given object array
    for(var i = 0; i<dat.length; i++){
        //For loop to iterate over the properties (key
        for (var key in dat[i]) {
            //Check if the given property is the direc
            //and not an inherited one
            if (dat[i].hasOwnProperty(key) == true) {
                //Check if the potential key has a val
                //function
                if (dat[i][key] === val){
                    //Return the key
                    return key
                }
```

<u>Layout 3</u>: *Circular Bar chart – Displays the ranking of the states, ordered by the rate of the chosen crime category (Property or Violent)*

(a)



(b)

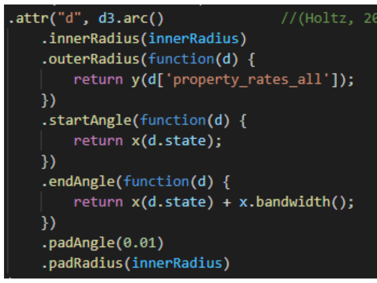


The above two images illustrate the ranking of the states when its ranked as per the (a) Property crime rate and (b) Violent crime rate.

Since the US has a total of 50 states and 1 federal district (Washington D.C.), displaying the ranking using a lollipop chart or bar chart would not be ideal as the bars/lollipops would appear close to each other and would look crowded. It would be hard to make observations if the dashboard used a bar chart or lollipop chart to display the ranking. Therefore, a circular bar plot was used to show the ranking.

D3's scaleBand() is used as the scale for the x-axis, in which the range goes from 0 to (2*PI) [PI = 3.14…..]. To ensure a complete circle is formed instead of a semi-circle, the range of the x-axis goes till (2*PI) (Holtz, 2022). The y-axis uses D3's scaleRadial() instead of the usual scaleLinear() because the scaleLinear() tends to exaggerate some of the high values present in the circular bar plot (Holtz, 2022).

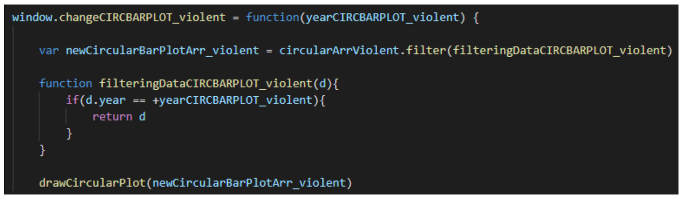
The SVG path element is used for creating the circular bar plot and all the bars present in the layout belong to the defined class "circular_property" or "circular_violent", depending on which crime category was selected. For a donut/pie chart, the arc() function usually computes the startAngle, endAngle, etc. automatically based on the input passed for generating the pie/donut chart. However, for a circular bar plot the code has manually sets the value of each of the properties of the arc() function. The following logic was used for achieving this. The code sets the inner radius value to a fixed one: 90, whereas the outer radius is generated using the values present under the "property_rates_all" or "violent_rates_all" keys (The Property or Violent crime rates recorded for each state in a particular year). Because the layout already shows the proportion (using the recorded total values) of the total number of crimes committed under each of the subcategories in the form of a donut chart, the recorded rates are used instead of the total values recorded for each major crime category (Property or Violent). Hence, the circular bar plot ranks the states as per the "Property" or "Violent" crime rates recorded for a given year (taken from the slider). The arc's start and end angles are determined



```
.attr("d", d3.arc()                    //(Holtz, 20
    .innerRadius(innerRadius)
    .outerRadius(function(d) {
        return y(d['property_rates_all']);
    })
    .startAngle(function(d) {
        return x(d.state);
    })
    .endAngle(function(d) {
        return x(d.state) + x.bandwidth();
    })
    .padAngle(0.01)
    .padRadius(innerRadius)
```

by the state and the x-axis scale, which is created using scaleBand(), as previously explained. Additionally, the bandwidth() of the x-axis is also used for computing the arc's end angle. A minimal amount of padding is put between the bars using the padAngle() function to ensure they don't appear too close to each other. The labels are added accordingly, and its position are set dynamically using the "transform" attribute. Counters are used to ensure multiple instances of the labels aren't created as the current year changes (using the slider). If the counter value is 1, then don't add any new labels, just move the labels accordingly. However, if the counter is 0, then add the labels to the chart. The state names serve as the labels.

Two separate global functions, callCircularBarplot_violent() and callCircularBarplot_property() are defined. Depending on which crime category is chosen by the user initially, one of these global functions will be executed.

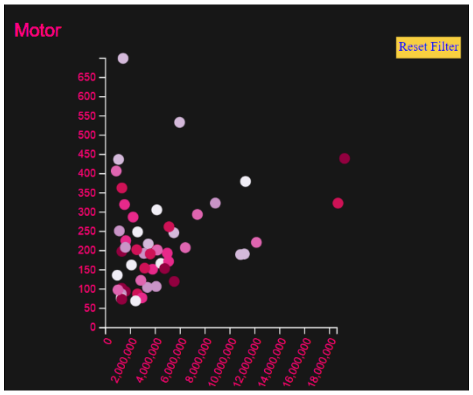


```
window.changeCIRCBARPLOT_violent = function(yearCIRCBARPLOT_violent) {

    var newCircularBarPlotArr_violent = circularArrViolent.filter(filteringDataCIRCBARPLOT_violent)

    function filteringDataCIRCBARPLOT_violent(d){
        if(d.year == +yearCIRCBARPLOT_violent){
            return d
        }
    }

    drawCircularPlot(newCircularBarPlotArr_violent)
```

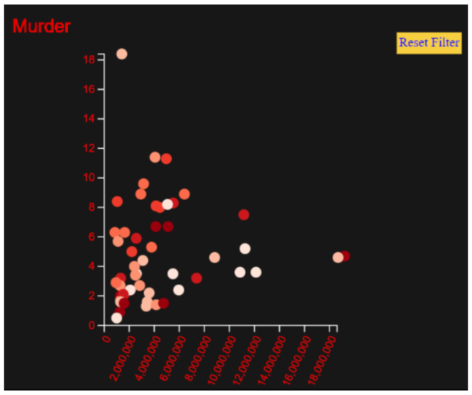
For example, if the user selects the "Property" crime category, the callCircularBarplot_property() global function will be executed. These functions create the SVG object for the circular bar plot and generates it for the year 1965 (the dashboard by default shows statistics for the year 1965). If the user changes the year value using the slider, the function changeCIRCBARPLOT_violent() or changeCIRCBARPLOT_property() is called. Within this function, only the values recorded for the selected year will be extracted out (using the filter() function) and the local function used for generating the circular bar plot will be called while passing the newly created object array as an argument. This local function will update the layout and the positions of the labels. Additionally, the circular bar plot makes use of the transition(), duration(), and exit() functions to ensure the bars animate smoothly while updating.

<u>Layout 4</u>: *Scatter plot – Shows the correlation between a crime subcategory and the overall population*

(a)



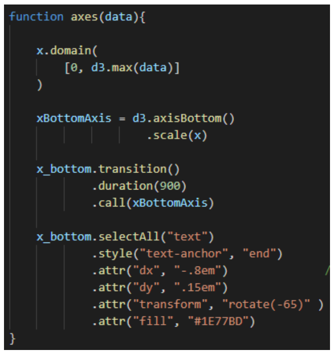
(b)



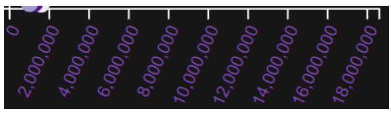
The above two images illustrate the correlation between a specific subcategory ((a) Motor [Property] and (b) Murder [Violent]) rate and the population of US for a particular year.

A scatter plot was used to see if there was a relationship between a particular crime subcategory and the population. This type of layout is best amongst the other to check for relationships.

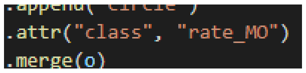
The scaleLinear() function was used for both the x and y axes, in which the ranges were generated as per the width and height (calculated by the code based on defined margin values) respectively. However, the domains are not manually set initially. This is done to ensure the domains of the x and y axes are dynamically set. The code snippet on the right shows how the domains are dynamically set. The logic for dynamically setting the domains of both the axes is as follows. Two functions axes() and updateYAxis() are defined for the x and y axes, respectively. The domain is set as per the maximum value present in the argument provided as an input to the function. To acquire the maximum value, the code uses D3's max() function and the domain begins from 0 onwards. As the domain is set, the values on the axes transitions (using transition() and duration()) and get updated. The same logic is used for updating the y-axis. Additionally, since some of the values on the x-axis are large, the code uses the transform attribute to rotate these values so that they appear in a slanted manner. To the axes() function, the code passes an array containing the total population values for each of the states and federal districts, to generate the values on the x-axis. Similarly, an array containing the rates of a certain crime subcategory are passed to the updateYAxis() function.



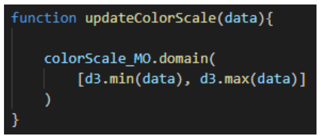
```
function axes(data){

    x.domain(
        [0, d3.max(data)]
    )

    xBottomAxis = d3.axisBottom()
                    .scale(x)

    x_bottom.transition()
            .duration(900)
            .call(xBottomAxis)

    x_bottom.selectAll("text")
            .style("text-anchor", "end")
            .attr("dx", "-.8em")
            .attr("dy", ".15em")
            .attr("transform", "rotate(-65)" )
            .attr("fill", "#1E77BD")
}
```



The scatter plot markers generated under each of the scatter plots belong to classes. For example, consider the scatter plot showing the correlation between the population and motor rate for a particular year. The markers generated for this plot, all belong to the "rate_MO" class. This was set using D3 and the attribute "class". Each of the markers were created using the "circle" SVG element and its x and y coordinates were set using the population and motor rate values (that is being currently read), respectively. Whenever there is an updation in the dashboard, the markers automatically transition to their new position using the transition(), duration() and exit() functions, to ensure smooth animation. As seen in the above two examples of the scatter plots, the markers appear in different colors. This is because the code defines an ordinal scale of colors, where the domain is dynamically set using the similar logic explained for the automatic updation of the two axes. The array containing the rates of a certain crime subcategory (passed also to the updateYAxis() function) is passed to the function used for automatically updating the domain of the color scale, i.e., the updateColorScale() function. the below D3 color schemes were used for each of the crime subcategories:



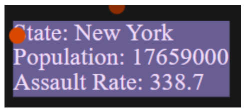
```
.append("circle")
.attr("class", "rate_MO")
.merge(o)
```

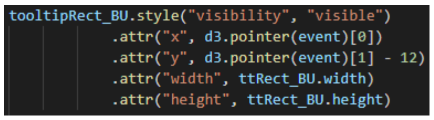


```
function updateColorScale(data){

    colorScale_MO.domain(
        [d3.min(data), d3.max(data)]
    )
}
```

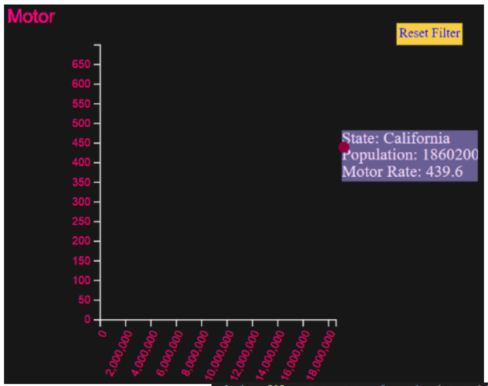
- Burglary: d3.schemeBlues
- Larceny: d3.schemeGreys
- Motor: d3.schemePuRd
- Assault: d3.schemeOranges
- Murder: d3.schemeReds
- Rape: d3.schemeGreens
- Robbery: d3.schemePurples

The image on the right shows an example of what happens when the mouse hovers over any one of the markers in the scatter plot. Similar to the map and donut chart layouts, each of the scatter plots have tooltips defined for the SVG text and rectangle element. The width and height of the SVG rectangle is defined as per the values generated by the BBox() function. However, this time the x and y coordinates of the SVG rectangle are not set as per the BBox() values, as the rectangle was not appearing in the desired location. To fix this, the code sets the coordinates of the rectangle based on the position of the mouse pointer.

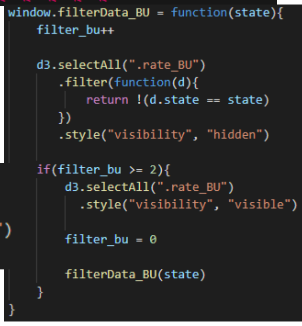




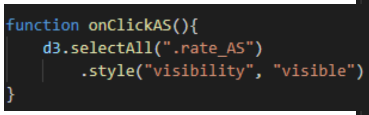
```
tooltipRect_BU.style("visibility", "visible")
            .attr("x", d3.pointer(event)[0])
            .attr("y", d3.pointer(event)[1] - 12)
            .attr("width", ttRect_BU.width)
            .attr("height", ttRect_BU.height)
```



The user has the choice to view the marker related to a state of their choice. By clicking on a state, for example, California, the dashboard only shows the markers (on all scatter plots for a particular major crime category – Property or Violent) that are related to California and the remaining ones are hidden using the "visibility" attribute. The global functions related to the scatter plots displayed on the dashboard are called, which is used for showing the desired marker on the plot. The code selects all elements belonging to the classes defined for each of the scatter plots and filters out the data for everything except the chosen state. As mentioned above, the "visibility" attribute is set to "hidden" for these filtered out values. The code initializes counters to keep track of the number of times the functions are called. The user can again select a different state of their choice, and the counter gets incremented to 2. As soon as the user clicks another state and the if statement shown on the code snippet gets triggered, the "visibility" attribute is reset to "visible", and the same function is called again to show only the marker of the chosen state. Additionally, the counter is also reset back to 0. If the user wishes to view all the markers back again, they can do so by clicking on the yellow "Reset Filter" button. This button selects all markers belonging to specific classes and resets their "visibility" attribute to "visible". This is done to ensure the chart can update to show the next chosen state's related marker.



```
window.filterData_BU = function(state){
    filter_bu++

    d3.selectAll(".rate_BU")
       .filter(function(d){
           return !(d.state == state)
       })
       .style("visibility", "hidden")

    if(filter_bu >= 2){
        d3.selectAll(".rate_BU")
          .style("visibility", "visible")

        filter_bu = 0

        filterData_BU(state)
    }
}
```

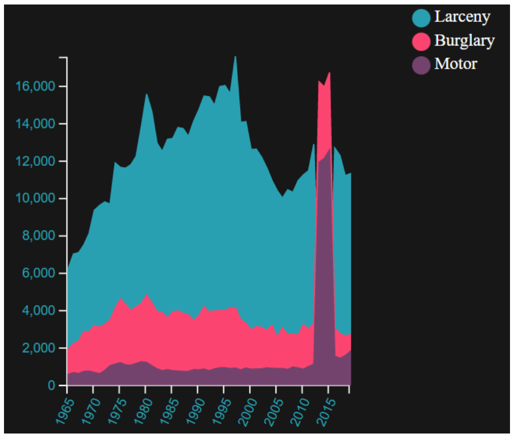


```
function onClickAS(){
    d3.selectAll(".rate_AS")
        .style("visibility", "visible")
}
```

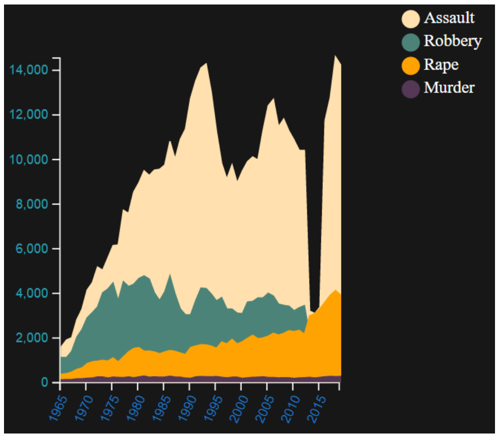
Similar to some of the above layouts, the data used for generating the scatter plot are the values recorded for the chosen year (taken from the slider).

Layout 5: *Area Chart – Displays the evolution of crime subcategories over the course of 55 years for a selected country*

(a)

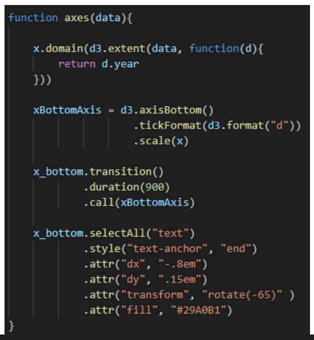


(b)



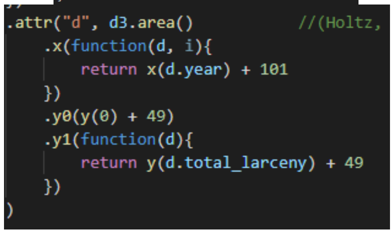
To show the trend of each crime subcategory, the area chart was used. This was chosen over the line chart as the area under the line help to infer certain patterns that can help in understanding the trend.

It can be seen from the above two images that different years appear on the x-axis. To achieve this, the code makes use of the scaleTime() function to generate a time scale for the x-axis. The domain is set using D3's extent() and the different years present in the dataset (1965-2019). The y-axis is set using the usual scaleLinear(). Using the above discussed logics, the y-axis is automatically updated in the same way. The total values of all the Property subcategories (Burglary, Larceny, etc.) or the total values of all the Violent subcategories (Assault, Robbery, etc.) are stored in an array. This array is fed to the function used for automatically updating the y-axis. The dataset fed to the function used for generating the area chart is sent as an argument to the function used for dynamically updating the x-axis. As discussed, the domain is set using D3's extent() function and the years present in the input data. Transitions are used for the smooth transitioning of the y and x axes.



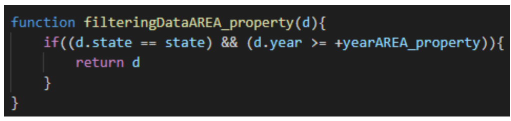
```
function axes(data){

    x.domain(d3.extent(data, function(d){
        return d.year
    }))

    xBottomAxis = d3.axisBottom()
                    .tickFormat(d3.format("d"))
                    .scale(x)

    x_bottom.transition()
            .duration(900)
            .call(xBottomAxis)

    x_bottom.selectAll("text")
            .style("text-anchor", "end")
            .attr("dx", "-.8em")
            .attr("dy", ".15em")
            .attr("transform", "rotate(-65)" )
            .attr("fill", "#29A0B1")
}
```

The code snippet on the right shows how the area under the line (for a specific subcategory) was calculated. Since the x-axis displays the different year values, to the x() property of D3's area() function, the code passes the year value to the defined x-axis scale. The result serves as the value for the x() property. The y0 property passes 0 to the y-axis scale so that area is generate accurately and the chart is positioned appropriately on the x-axis and no gap is there between the base of the chart and the x-axis. Whereas, for the y1() property, the total value of a subcategory is passed to the y-axis scale. Like the above layouts, transition(), exit(), and duration() are used for ensuring the updating of the chart is done smoothly and not suddenly.

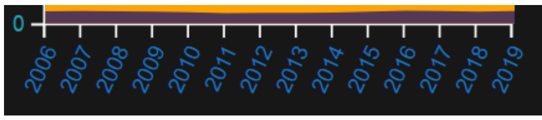


```
.attr("d", d3.area()              //(Holtz,
    .x(function(d, i){
        return x(d.year) + 101
    })
    .y0(y(0) + 49)
    .y1(function(d){
        return y(d.total_larceny) + 49
    })
)
```

To get the effect shown in the area chart screenshots above, the area chart of each subcategory was computed separately one after the other and each of these charts belong to different classes so that there isn't any confusion while accessing the elements of a certain class. (Shown in the JavaScript code files). The area chart shows only the trend for a particular country that was selected on the map. The code snippet on the right shows how the data for the chart is extracted. In addition to extracting the records collected for a certain state, records collected on the chosen year and the upcoming years as well are extracted out. This is done so that the dashboard can show the trend over time. The screenshot on the right shows how the x-axis appears when the slider is on the year 2006.



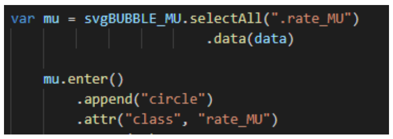
```
function filteringDataAREA_property(d){
    if((d.state == state) && (d.year >= +yearAREA_property)){
        return d
    }
}
```



Design Patterns, MVP, Classes, and Objects:

The two most commonly used OOP concepts by the dashboard were Encapsulation and Inheritance.

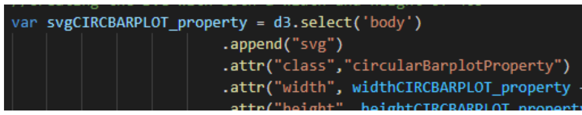
Use of Encapsulation:



```
var mu = svgBUBBLE_MU.selectAll(".rate_MU")
                        .data(data)

    mu.enter()
        .append("circle")
        .attr("class", "rate_MU")
```

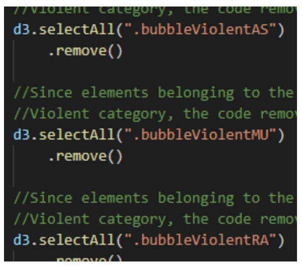
Nearly all the charts present on the dashboard wrapped their elements within a class. This was not done by explicitly writing the keyword "class" but was done using D3. The code snippet on the right shows how this was done. The SVG object selects all the elements belonging to the "rate_MU" class, and these elements were accessed by using the dot operator. This operator is generally used for accessing classes. While creating the desired elements and appending the SVG circle elements, the code snippet above shows that the class called "rate_MU" is created by using the attribute "class". By doing this, all elements are now wrapped under the "rate_MU" class and they inherit the properties of the class.

Use of Inheritance:



```
var svgCIRCBARPLOT_property = d3.select('body')
                                .append("svg")
                                .attr("class","circularBarplotProperty")
                                .attr("width", widthCIRCBARPLOT_property
```

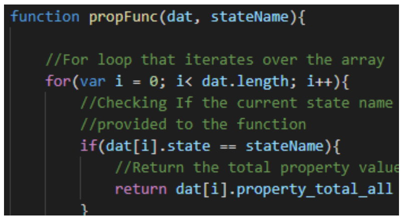
All the layouts were wrapped in a super class that is created while generating the SVG object. As shown in the code snippet on the right, the class "circularBarplotProperty" serves as the superclass. All the elements created under this SVG will inherit the properties of the "circularBarplotProperty" superclass such as its defined "transform" attribute, which will position the chart/layout accordingly.



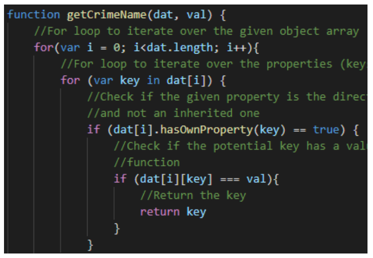
```
d3.selectAll(".bubbleViolentAS")
    .remove()

//Since elements belonging to the
//Violent category, the code remov
d3.selectAll(".bubbleViolentMU")
    .remove()

//Since elements belonging to the
//Violent category, the code remov
d3.selectAll(".bubbleViolentRA")
```

The classes defined in each of the JavaScript files don't use constructors. The use of classes helped ease multiple processes such as showing the marker of a certain country based on whichever state was clicked on, displaying specific layouts for a chosen crime category (Property or Violent), etc. while developing the dashboard.

Some object factories were created, and the logics were explained above. The below code snippets show some object factories that were created in the JavaScript files to return specific objects.



```
function propFunc(dat, stateName){

    //For loop that iterates over the array
    for(var i = 0; i< dat.length; i++){
        //Checking If the current state name
        //provided to the function
        if(dat[i].state == stateName){
            //Return the total property value
            return dat[i].property_total_all
        }
```



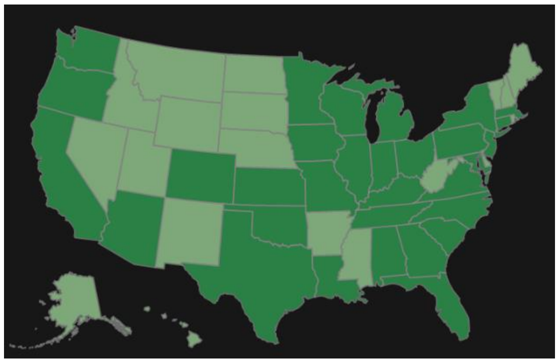
```
function getCrimeName(dat, val) {
    //For loop to iterate over the given object array
    for(var i = 0; i<dat.length; i++){
        //For loop to iterate over the properties (key
        for (var key in dat[i]) {
            //Check if the given property is the direc
            //and not an inherited one
            if (dat[i].hasOwnProperty(key) == true) {
                //Check if the potential key has a val
                //function
                if (dat[i][key] === val){
                    //Return the key
                    return key
                }
            }
```

The dashboard makes use of MVP's as well. In most of the layouts, for example, the scatter plot, donut chart, etc., the data was modified (using the filter function) to only extract the data for a particular year or country or a range of years. This was done so that the dashboard only displays the required data, that will help gain more insights and be intuitive.

Hypotheses and Conclusion:

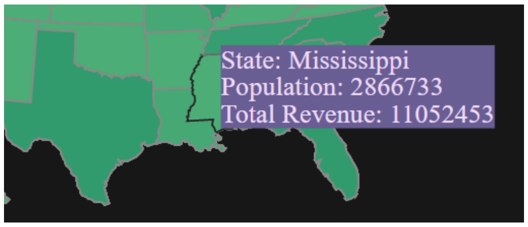
The below are some hypotheses and conclusions made while observing the dashboard.

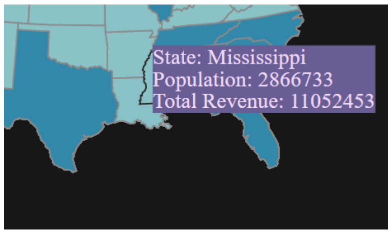
H0: The crime rate might be more in either one of the regions of USA – Western, Northeast, Midwest, or Southern Region.



<u>Conclusion:</u> Location of a state doesn't play a major role in the increase of crime rate. For example, the below map shows how the US map appears for the Property crime category in the year 1965. It can be seen that the property crime rate is high in most of the states that are present in each of the above mentioned four regions. A similar information was observed in the Violent choropleth map. Hence, location does not matter for the crime rate. We can conclude that Hypotheses H0 failed.

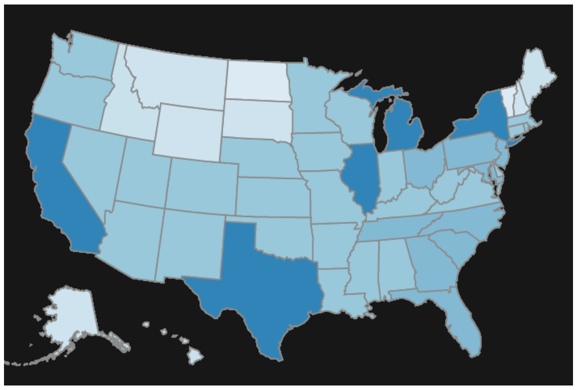
H1: Poverty is known to play a major role in the increase in the crime rate





<u>Conclusion:</u> The state of Mississippi is said to be the poorest state in USA. However, as seen from the above two images, states like Texas and Florida have a higher crime rate as compared to Mississippi. Hence, in this case Finance is not related to the increase in crime rate. We can conclude that Hypotheses H1 failed.

H2: Highly populous states have a higher crime rate



<u>Conclusion:</u> In spite of showing a weak correlation in the scatter plots, from the map, we can infer that highly populous states such as California, New York, etc. have higher crime rates as compared to the other states. Hence, we can conclude that H2 is true.

Limitations and Improvements:

1. The scale for the area chart does not automatically update when changing the years using the slider. The user has to manually click on the state to see the change in the area chart. This can be potentially improved by accessing the class that wraps the area hart elements and modifying the axes using a global function.
2. Drugs are also known to play a major role in the increase of crime. However, the online available datasets don’t have data before the year 2000. The datasets have data recorded from the 2000s onwards.
3. The finance dataset used has only data collected from 1992 to 2019. More research is to be conducted to find datasets having the required information for the missing years.

References:

1. Bart, A.C., Whitcomb, R., Kafura, D., Shaffer, C.A. and Tilevich, E. (2017). Computing with CORGIS. *ACM Inroads*, 8(2), pp.66–72.

2. Holtz, Y. (2022). *Basic circular barplot in d3.js*. [online] D3-graph-gallery.com. Available at: https://d3-graph-gallery.com/graph/circular_barplot_basic.html
3.