Visualization mini-project 1 Report

Debesh Mohanty(111973815)

February 2019

1. Introduction

This mini project is based on visualization of North Carolina's crime data from 1980. This data consists of 630 data points and ten attributes. The attributes are crime rate, probability of arrest, probability of conviction, probability of sentence, police per capita, average sentence days, tax revenue per capita, population density, percentage of minorities, diversity of people and percentage of young males. The visualization includes bar graphs and pie charts. I have also created random edges between hundred data points for five attributes to construct a force directed layout.

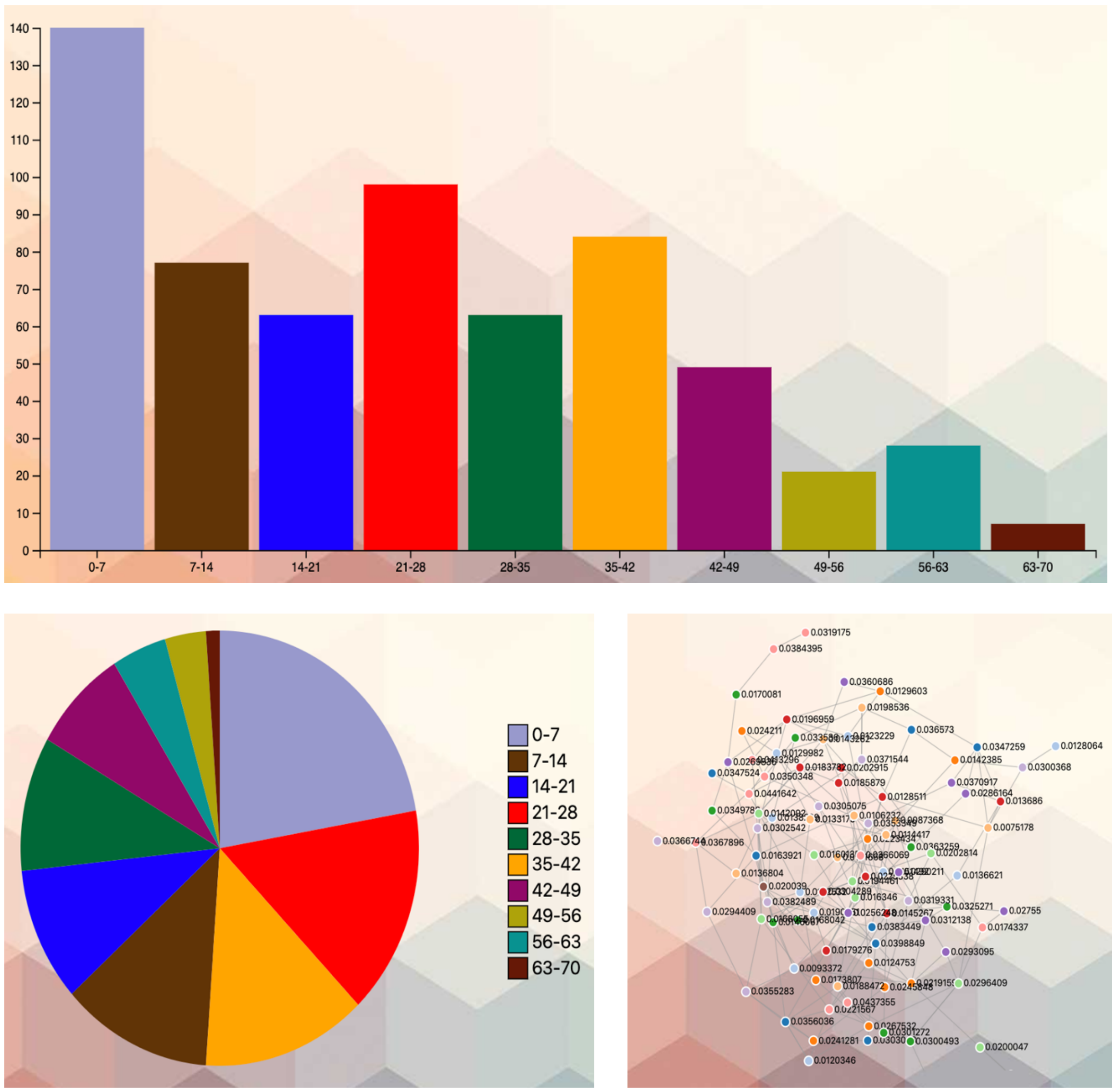
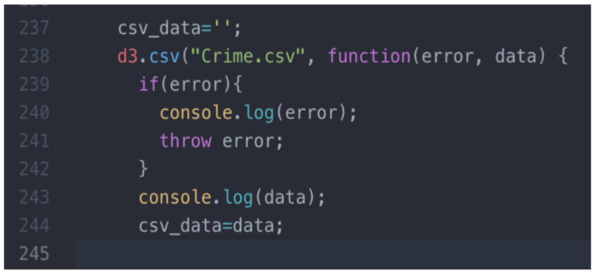


Fig1: Bar graph, Pie Chart and Force directed layout

2. Code Implementation

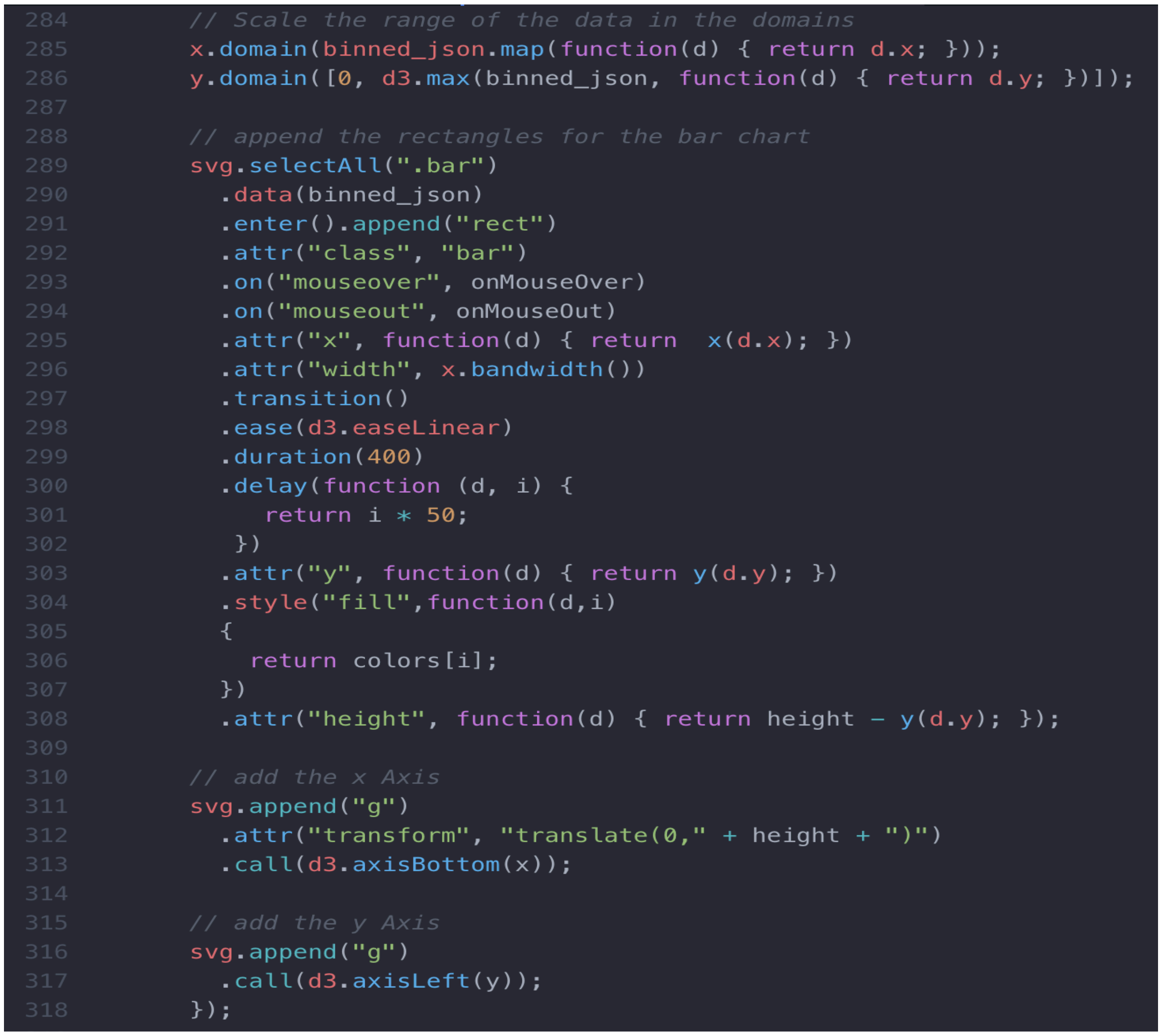
The entire web page has been designed using HTML, JavaScript, jQuery and D3.js. I provided a drop-down menu to select the attribute which the user wants to visualize. Three radio buttons select either bar graph or pie chart or force directed layout. The basic flow of this web page is to first read the data provided in a csv file and then use D3.js library to draw visualizations.



```
    csv_data='';
    d3.csv("Crime.csv", function(error, data) {
      if(error){
        console.log(error);
        throw error;
      }
      console.log(data);
      csv_data=data;
```

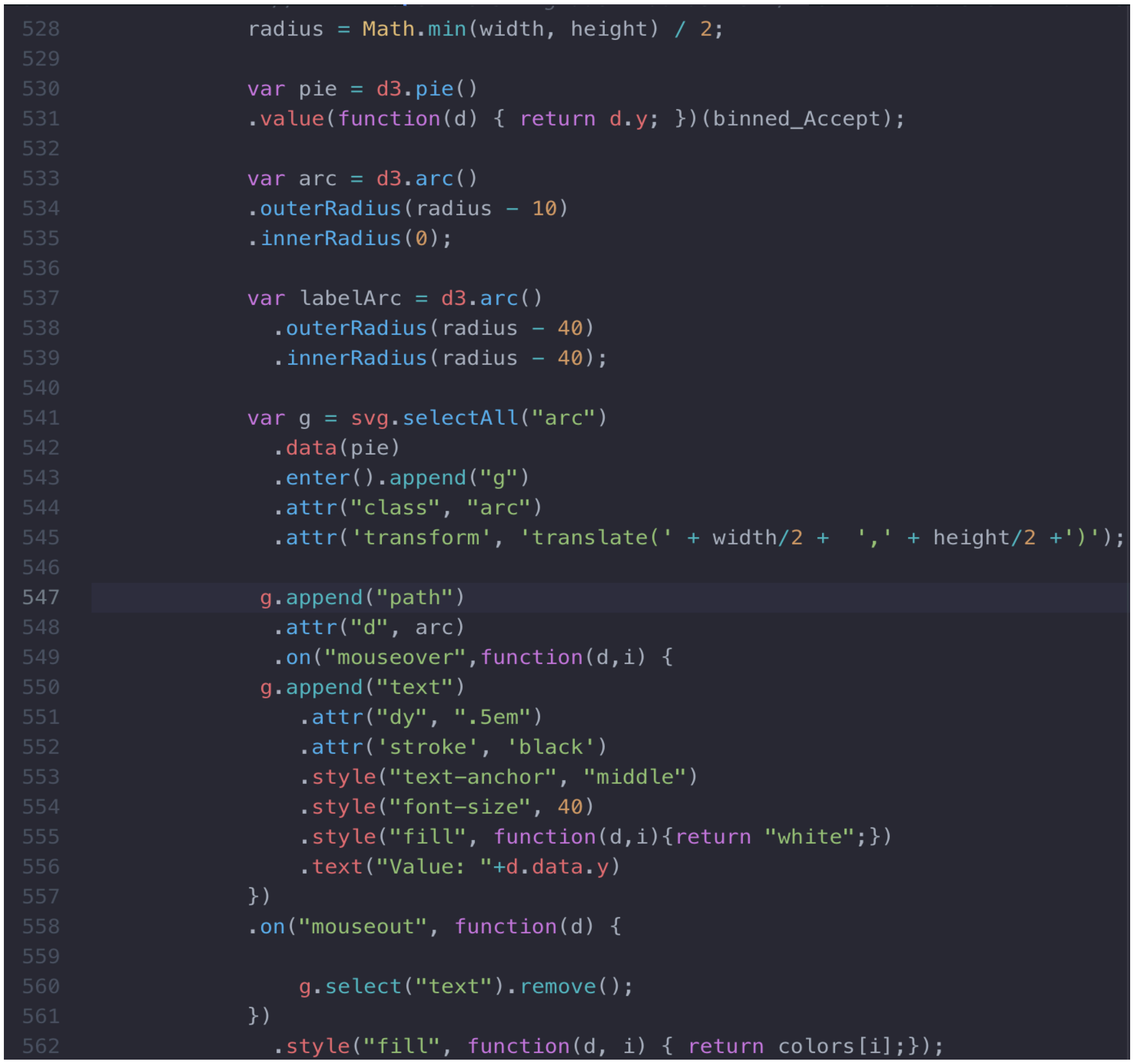
Fig2: Reading CSV file

For drawing the bar graph we need to first create a SVG layout in the DOM element. And then append to that SVG rectangles and axes to make the bar graph. I define x and y scales from the data and then draw rectangles. I then also add a transition(animation) effect in the drawing of the rectangles and finally draw the x and y axes.



```
        // Scale the range of the data in the domains
        x.domain(binned_json.map(function(d) { return d.x; }));
        y.domain([0, d3.max(binned_json, function(d) { return d.y; })]);

        // append the rectangles for the bar chart
        svg.selectAll(".bar")
          .data(binned_json)
          .enter().append("rect")
          .attr("class", "bar")
          .on("mouseover", onMouseOver)
          .on("mouseout", onMouseOut)
          .attr("x", function(d) { return  x(d.x); })
          .attr("width", x.bandwidth())
          .transition()
          .ease(d3.easeLinear)
          .duration(400)
          .delay(function (d, i) {
            return i * 50;
           })
          .attr("y", function(d) { return y(d.y); })
          .style("fill",function(d,i)
          {
            return colors[i];
          })
          .attr("height", function(d) { return height - y(d.y); });

        // add the x Axis
        svg.append("g")
          .attr("transform", "translate(0," + height + ")")
          .call(d3.axisBottom(x));

        // add the y Axis
        svg.append("g")
          .call(d3.axisLeft(y));
        });
```

Fig 3: Drawing bar graph



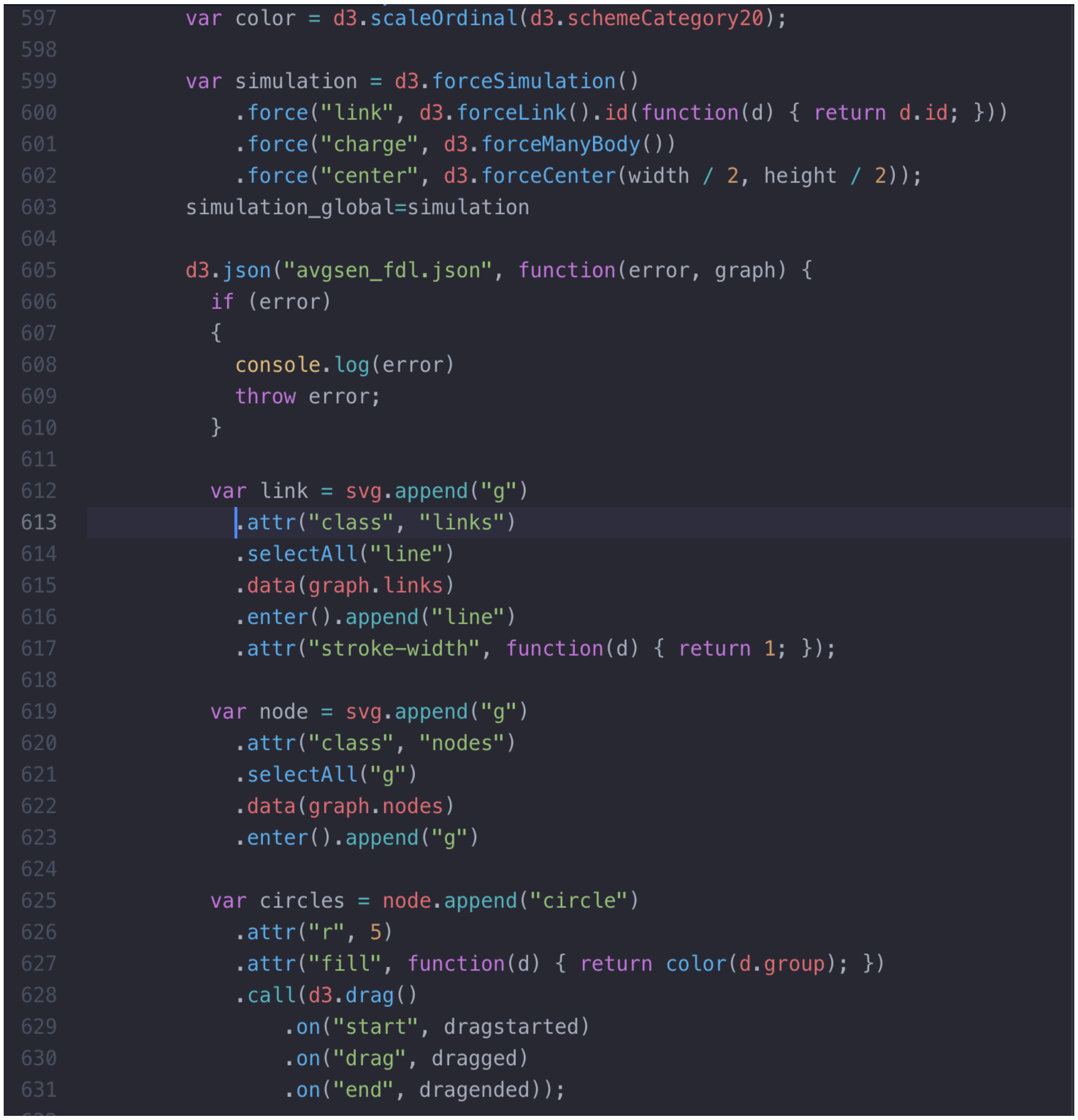
```
radius = Math.min(width, height) / 2;

var pie = d3.pie()
.value(function(d) { return d.y; })(binned_Accept);

var arc = d3.arc()
.outerRadius(radius - 10)
.innerRadius(0);

var labelArc = d3.arc()
  .outerRadius(radius - 40)
  .innerRadius(radius - 40);

var g = svg.selectAll("arc")
  .data(pie)
  .enter().append("g")
  .attr("class", "arc")
  .attr('transform', 'translate(' + width/2 +  ',' + height/2 +')');

 g.append("path")
  .attr("d", arc)
  .on("mouseover",function(d,i) {
 g.append("text")
    .attr("dy", ".5em")
    .attr('stroke', 'black')
    .style("text-anchor", "middle")
    .style("font-size", 40)
    .style("fill", function(d,i){return "white";})
    .text("Value: "+d.data.y)
})
.on("mouseout", function(d) {

    g.select("text").remove();
})
  .style("fill", function(d, i) { return colors[i];});
```

Fig 4: Drawing pie chart

For drawing the pie chart I first calculated the radius of the pie chart. Since I am trying to draw the SVG in half of the area of the SVG layout I took the half of the minimum of height and width of the SVG layout as my radius. I then compute the angles of the arcs using the value computed from my datasets while drawing the bar chart. At last I append all the arcs with a mouseover event and a color legend. The mouseover event is defined so that I can display the value of an arc whenever the user hovers mouse over that arc. I also added so properties to the text such as font size, stroke color and text-anchor to properly align the text displaying values of the arc on mouse hover.

While developing the bar charts and pie charts I pass a variable bin_width the function which defines the bin size that should be shown on the visualization. This variable is controlled by a slider that I have added in the web page. If the slider moves right the bin width increases and if it moves left the bin width decreases. This slider functionality is not available for the force directed layout as I am not using bin width while constructing the force directed layout.

For developing the force directed layout I first created JSON files for the selected attributes using a python script which randomly picks up edges between randomly chosen hundred nodes(data points). After reading the JSON file I draw nodes and edges based on the random generation of data. And register three callback functions – dragstarted, drag and dragended for starting of the drag, drag transitions and transitions after dragging respectively. Inside these functions I use a simulation object that was previously defined. This simulation object is created from the d3.forceSimulation() API in the below code. This handles all the force related transitions inside these three callback functions.



```
        var color = d3.scaleOrdinal(d3.schemeCategory20);

        var simulation = d3.forceSimulation()
            .force("link", d3.forceLink().id(function(d) { return d.id; }))
            .force("charge", d3.forceManyBody())
            .force("center", d3.forceCenter(width / 2, height / 2));
        simulation_global=simulation

        d3.json("avgsen_fdl.json", function(error, graph) {
          if (error)
          {
            console.log(error)
            throw error;
          }

          var link = svg.append("g")
            .attr("class", "links")
            .selectAll("line")
            .data(graph.links)
            .enter().append("line")
            .attr("stroke-width", function(d) { return 1; });

          var node = svg.append("g")
            .attr("class", "nodes")
            .selectAll("g")
            .data(graph.nodes)
            .enter().append("g")

          var circles = node.append("circle")
            .attr("r", 5)
            .attr("fill", function(d) { return color(d.group); })
            .call(d3.drag()
                .on("start", dragstarted)
                .on("drag", dragged)
                .on("end", dragended));
```

Fig 5: Drawing force directed layout

Finally after adding code for all the visualization I rendered the UI by just adding a background image, adding some custom written CSS and using some bootstrap classes.