# CSE 564 Visualization
# Lab 1 Report

Dataset:
The dataset used for Lab 1 is a collection of different attributes that play a part in getting admission into the graduate program of a certain university. The dataset consists of 500 data points (rows) and 9 dimensions (columns) of which 1 dimension is the application ID while the other 9 are numeric dimensions like GRE score, TOEFL score, etc.

Implementation details:

1. Pick a variable and bin it into a fixed range (equi-width) of your choice
    a. I initially set the number of bins to be equal to 5. To calculate the bin size of the variables, I use the formula:

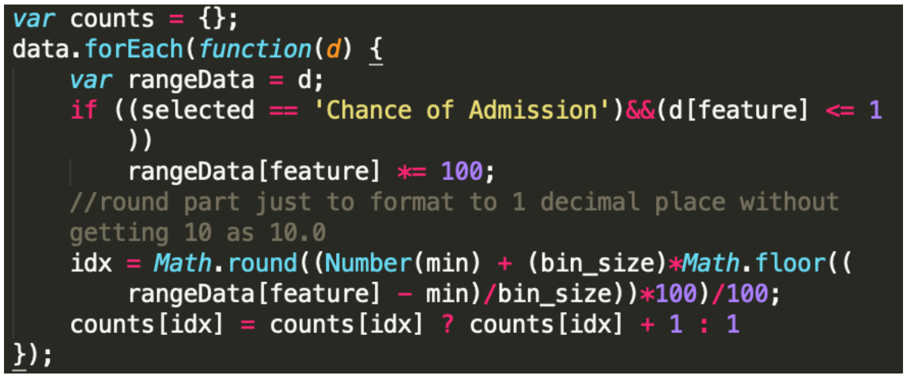
    $$bin\ size = \frac{\max(data) - \min(data)}{no. of\ bins}$$

    This will create equal sized (equi-width) bins of the entire range of the variable.

    b. I create a dictionary where each entry corresponds to a separate bucket. The key is the lower limit of the bucket and the value is the count for that bucket. To assign the data to either one of the buckets, I iterate through the data and add the data to an entry in the dictionary using the below formula for the key:

    $$key = \ floor(\frac{data[i] - \min(data)}{bin\ size}$$

    Here, data[i] corresponds to the current data being processed while iterating

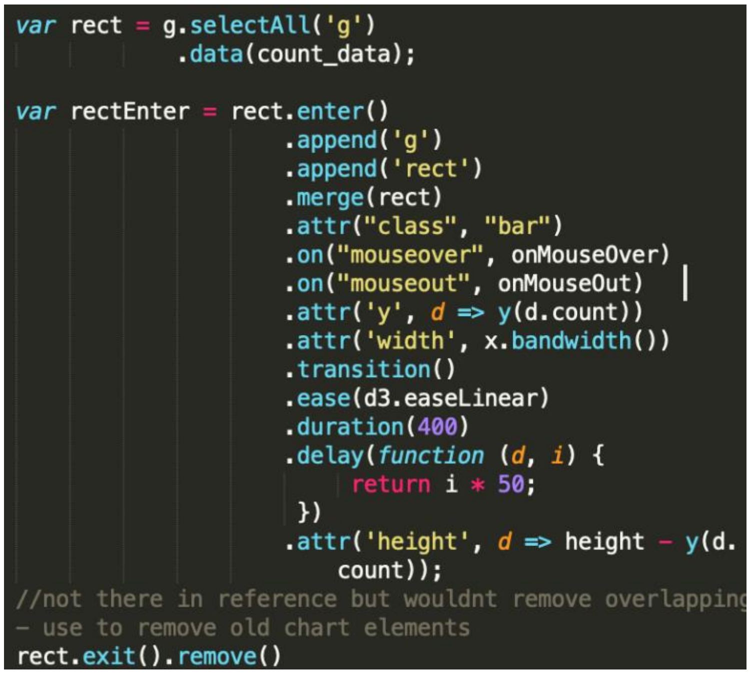


```
var counts = {};
data.forEach(function(d) {
    var rangeData = d;
    if ((selected == 'Chance of Admission')&&(d[feature] <= 1
        ))
        rangeData[feature] *= 100;
    //round part just to format to 1 decimal place without
    getting 10 as 10.0
    idx = Math.round((Number(min) + (bin_size)*Math.floor((
        rangeData[feature] - min)/bin_size))*100)/100;
    counts[idx] = counts[idx] ? counts[idx] + 1 : 1
});
```

2. Create a bar chart of the variable you picked in 1.
    a. In the next step, I create a bar chart of the GRE score where number of bins = 5.
    b. I start off by drawing rectangles corresponding to each bin in my object array. These <rect> elements are not already present in the DOM so I make use of the enter-update-exit paradigm wherein I put placeholders for the number of rectangles I required (obtained from the data) and appended the <rect> elements later. I added a transition to the drawing of <rect> for aesthetic effect.
    c. To position the elements, I created two scaling functions ‘x’ and ‘y’ to scale the data I was plotting appropriately so that it could be plotted in the correct

position with respect to the size of the <svg>. In addition, the height of each of the bars in the chart have to be calculated as $height(svg) - y(count\ of\ bar)$ because the origin of the svg lies in the upper left corner.

d. The x axis contains the range and y axis contains the count of each bucket.



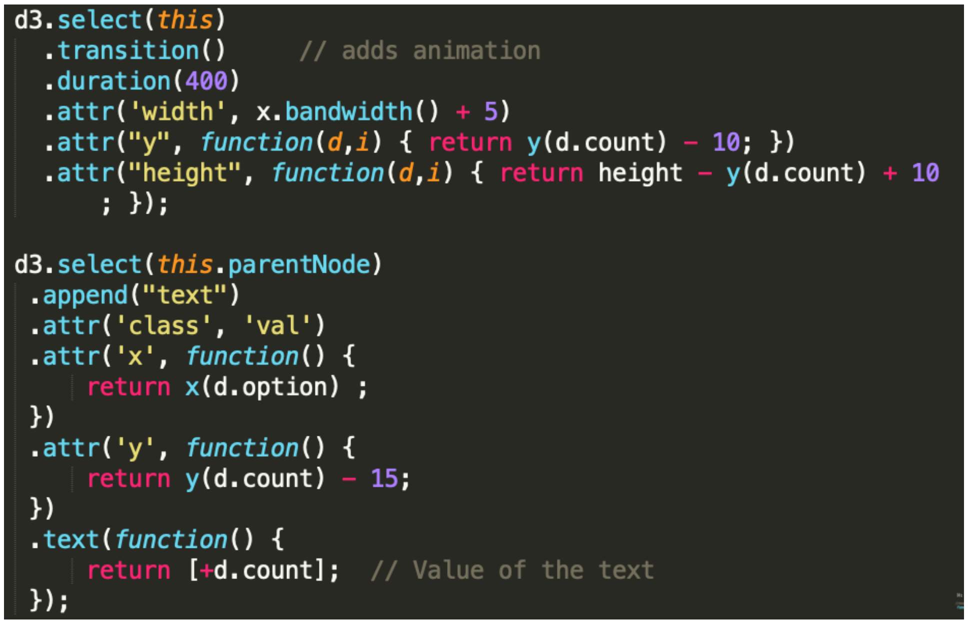
```
var rect = g.selectAll('g')
            .data(count_data);

var rectEnter = rect.enter()
                    .append('g')
                    .append('rect')
                    .merge(rect)
                    .attr("class", "bar")
                    .on("mouseover", onMouseOver)
                    .on("mouseout", onMouseOut)
                    .attr('y', d => y(d.count))
                    .attr('width', x.bandwidth())
                    .transition()
                    .ease(d3.easeLinear)
                    .duration(400)
                    .delay(function (d, i) {
                        return i * 50;
                    })
                    .attr('height', d => height - y(d.
                       count));
//not there in reference but wouldnt remove overlapping
- use to remove old chart elements
rect.exit().remove()
```

3. Using a menu, allow users to select a new variable and update chart
   a. With the help of a dropdown menu, I added the other 7 variables as options to choose a new dimension to display on the chart.
   b. To implement this, I put the entire drawing of my bar chart within a function called the update function. I also add three more functions function to the creation of my bar chart – the 'merge( )', 'exit( )' and 'remove( )' functions.
      i. The merge function merges the previous existing DOM elements to the newly created DOM elements (of the new dimension chosen).
      ii. The exit and remove function are used together to select all redundant DOM elements (from the previous data) and removes them.
   c. Selecting a new dimension from this dropdown menu calls this update function which takes the new dimension selected as one of its arguments and updates the bar chart for the new dimension.

4. Only on mouse-over display the value of the bar on top of the bar
   a. To implement this, I add an event listener function for 'mouseover' to every <rect> of the bar chart.
   b. When the function is called, I navigate to the <g> element that wraps every <rect> in order to append the text as <text> elements cannot be appended to <rect> elements.

    c. I append the <text> elements to the <g> element, positioning it just above the top of the bar using the ‘x’ and ‘y’ scaling functions and considering the upper left origin of the svg.
    d. I add another event listener function on ‘mouseout’ which removes the text when the mouse pointer is moved out of the area of the <rect>



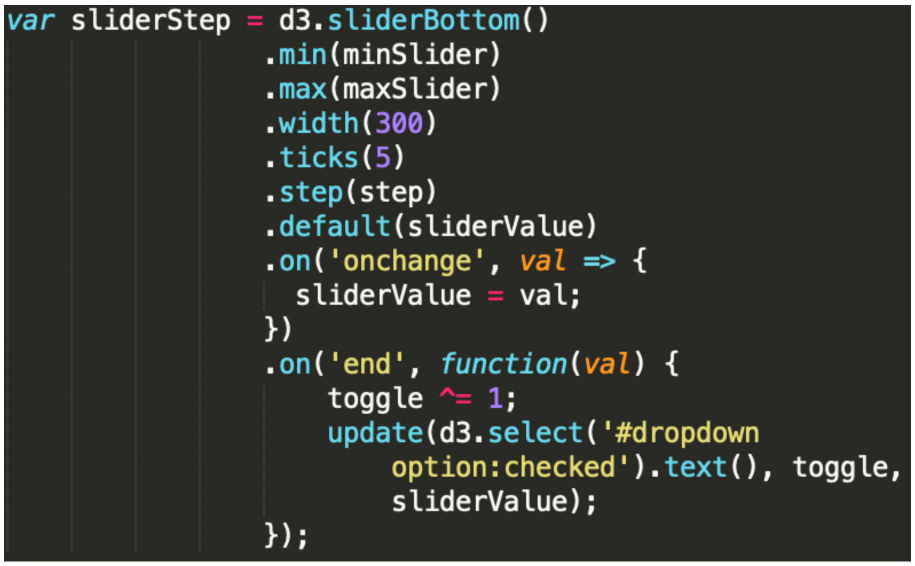
```
d3.select(this)
  .transition()     // adds animation
  .duration(400)
  .attr('width', x.bandwidth() + 5)
  .attr("y", function(d,i) { return y(d.count) - 10; })
  .attr("height", function(d,i) { return height - y(d.count) + 10
      ; });

d3.select(this.parentNode)
 .append("text")
 .attr('class', 'val')
 .attr('x', function() {
     return x(d.option) ;
 })
 .attr('y', function() {
     return y(d.count) - 15;
 })
 .text(function() {
     return [+d.count];  // Value of the text
 });
```

5. On mouse-over make the bar wider and higher to focus on it
    a. In the same event listener function for ‘mouseover’ as mentioned above, I increase the width and the height of the <rect> as well as change the ‘y’ attribute of the <rect> origin which changes its position with increase in height.
    b. In the same event listener function on ‘mouseout’ as mentioned above, I change the values of the ‘width’, ‘height’ and ‘y’ attributes of the <rect> back to its original position.

6. On mouse-click transform the bar chart into a pie chart (and back)
    a. Using the same data, we create a pie chart using D3’s pie( ) function to parse the data and get slice related information which we use to plot the arc (path) of each slice.
    b. When hovering over each individual slice of the pie chart, the slice is highlighted (changes color & becomes bigger) and displays information about the slice (bin). For aesthetic pleasure, I made a donut chart instead by changing the innerRadius
    c. The pie becomes visible to the viewer when they click over the bar chart. For this purpose, a event listener function on ‘click’ was created where it calls a function called update( ). This update function has an argument called ‘ toggle’ which indicates whether a bar chart or a pie chart is to be displayed which toggles on each click.

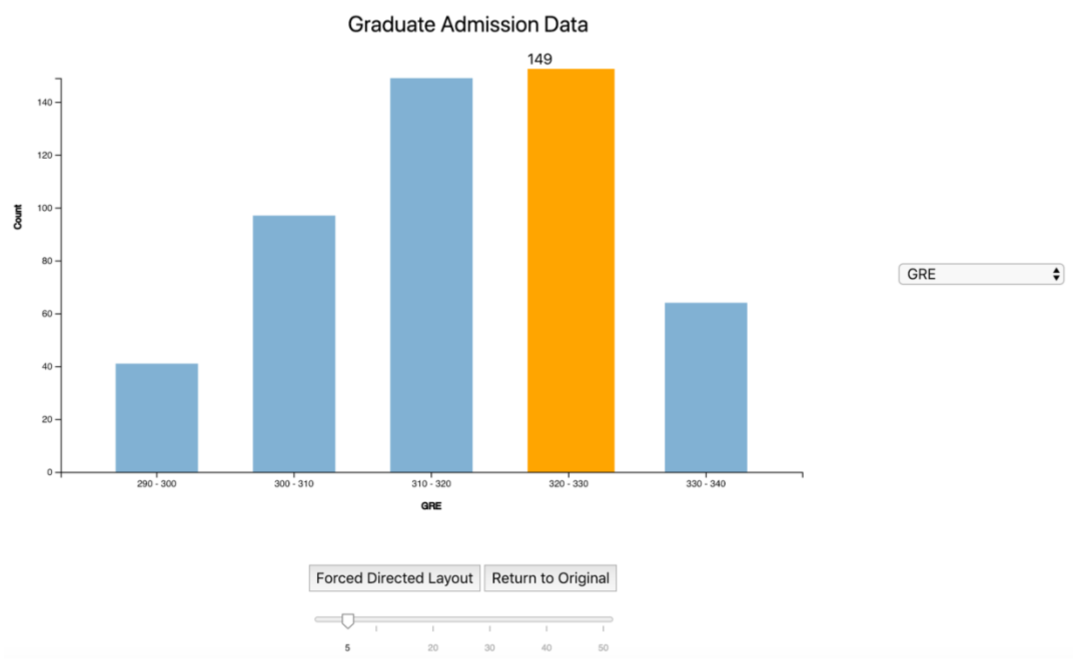
7. Mouse moves left (right) should decrease (increase) bin width/size
    a. To implement this, I used a slider. The position of the pointer on the slider corresponds to a particular value of the number of buckets.
    b. I added another argument to the update function indicating the number of buckets. I added an event listener function on ‘end’ i.e. release of the mouse, which will call this update function with the new value of the number of buckets. This will calculate the new bucket size and create the buckets once again as seen in 1.
    c. Note that since I have added the flexibility of using the slider to give number of buckets as any value from 1 to max number of buckets, in order to maintain the equal width of the buckets, the start and end limits of the buckets may be in decimal values.



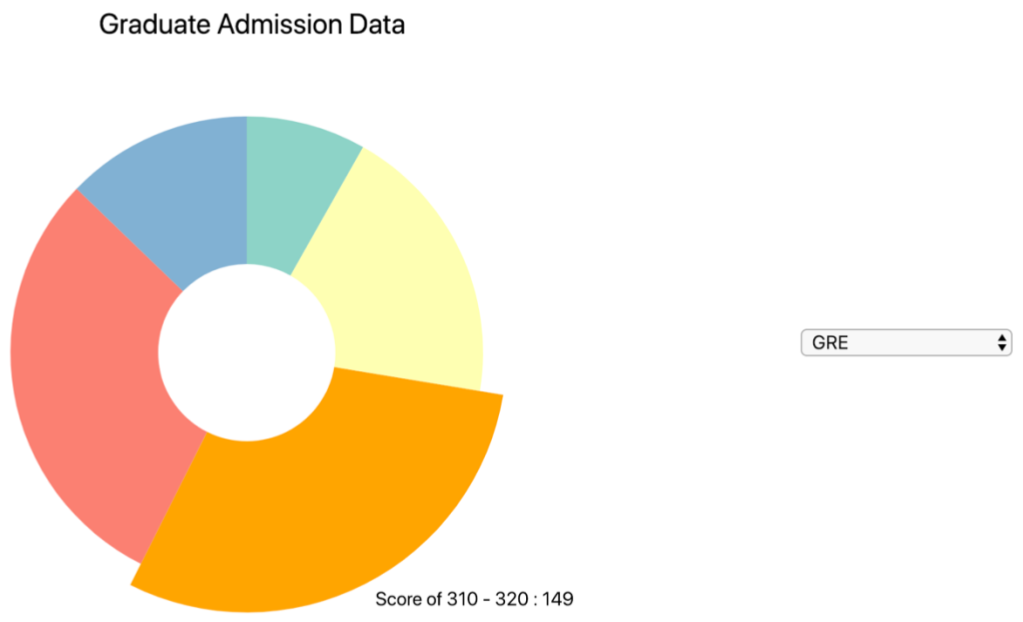
```
var sliderStep = d3.sliderBottom()
                .min(minSlider)
                .max(maxSlider)
                .width(300)
                .ticks(5)
                .step(step)
                .default(sliderValue)
                .on('onchange', val => {
                  sliderValue = val;
                })
                .on('end', function(val) {
                    toggle ^= 1;
                    update(d3.select('#dropdown
                        option:checked').text(), toggle,
                        sliderValue);
                });
```

8. Extra credit: Forced Directed Layout
    a. I created a force directed layout for a subset of elements of my dataset for my ‘GRE score’ dimension.
    b. This was created using D3’s force layout by setting up a force simulation. I added different forces to this simulation such as ‘forceCenter( )’ which sets up a center of gravity for the system and ‘forceManyBody( )’ which makes elements attract or repel each other.
    c. For every iteration of the simulation, a function called ticked is called which links each node (data point) to the <circle> element and updates their position.

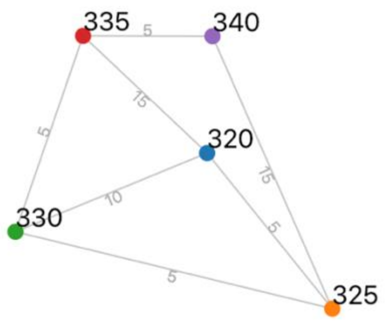
Illustration of program:



Graduate Admission Data
149
140
120
100
80
60
40
20
0
Count
290 - 300
300 - 310
310 - 320
320 - 330
330 - 340
GRE
GRE
Forced Directed Layout
Return to Original
5
20
30
40
50



Graduate Admission Data
GRE
Score of 310 - 320 : 149



335
340
320
330
325
5
15
5
10
15
5
5