CSE 564
Visualization and Visual Analytics

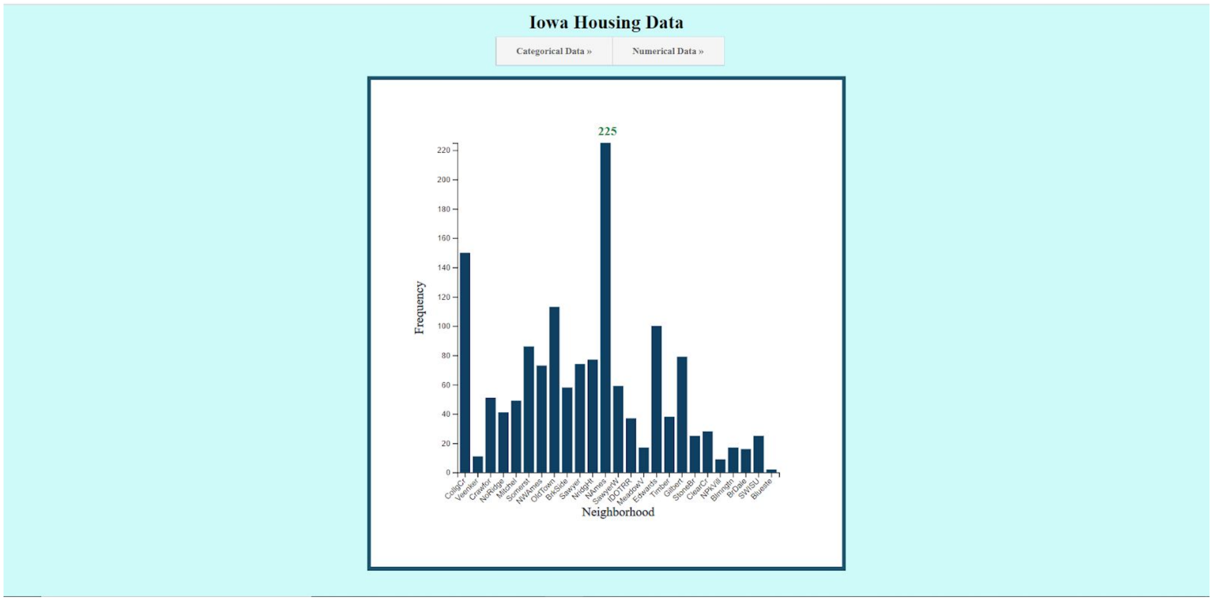
Mini Project #1

Pratik Mukund Velhal (112675099)

Project Overview:

This project visualizes csv data in the form of bar graphs and histograms for categorical and numerical data respectively. Augmenting visual features like tool-tip, dynamic bin size, and mouse hover animations have been added.

Demo: CSE 564:Visualization Project 1- Pratik Mukund Velhal

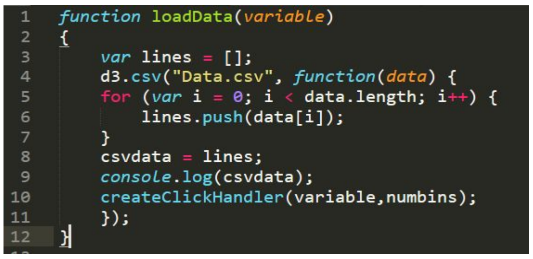


Data:

I have used the Ames Housing Dataset (**https://tinyurl.com/yzjolrx8**) available on Kaggle in this project. I had used this dataset previously for another data science project, and this dataset was chosen due to familiarity with all the variables. For our visualization, I have chosen a mix of 8 categorical and 8 numerical variables where each variable distinctly describes a unique characteristic of housing in Iowa. The dataset has 1460 rows of data items.

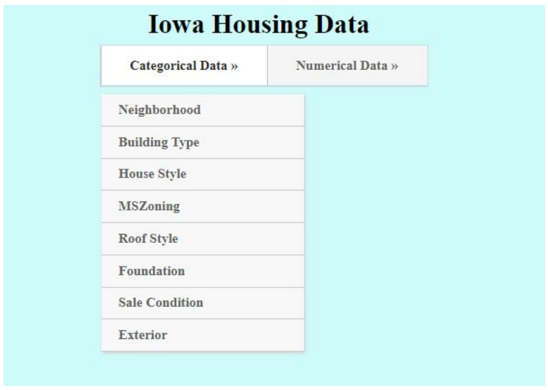
Implementation:

1. Reading data: Our input is in the form of a csv file. We are loading data from this file using d3.csv(). Data is loaded from file only on first call to variable plot, and thereafter is re-used to improve efficiency.



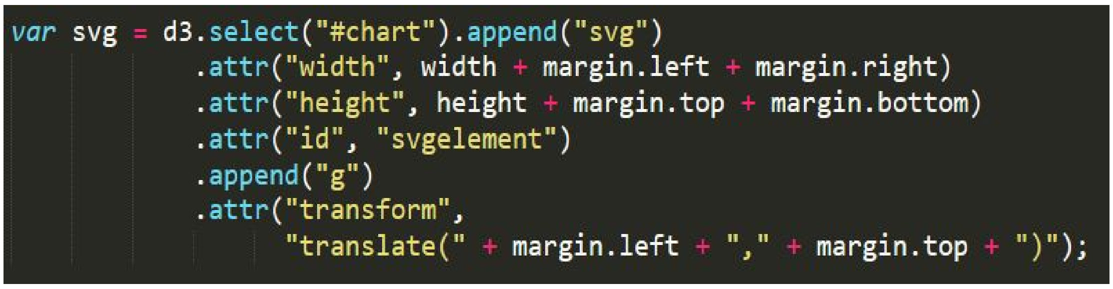
```
function loadData(variable)
{
    var lines = [];
    d3.csv("Data.csv", function(data) {
    for (var i = 0; i < data.length; i++) {
        lines.push(data[i]);
    }
    csvdata = lines;
    console.log(csvdata);
    createClickHandler(variable,numbins);
    });
}
```

2. Choosing variables: I have created two separate drop down menus for categorical and numerical variables respectively, and populated the variable names in the drop-down.



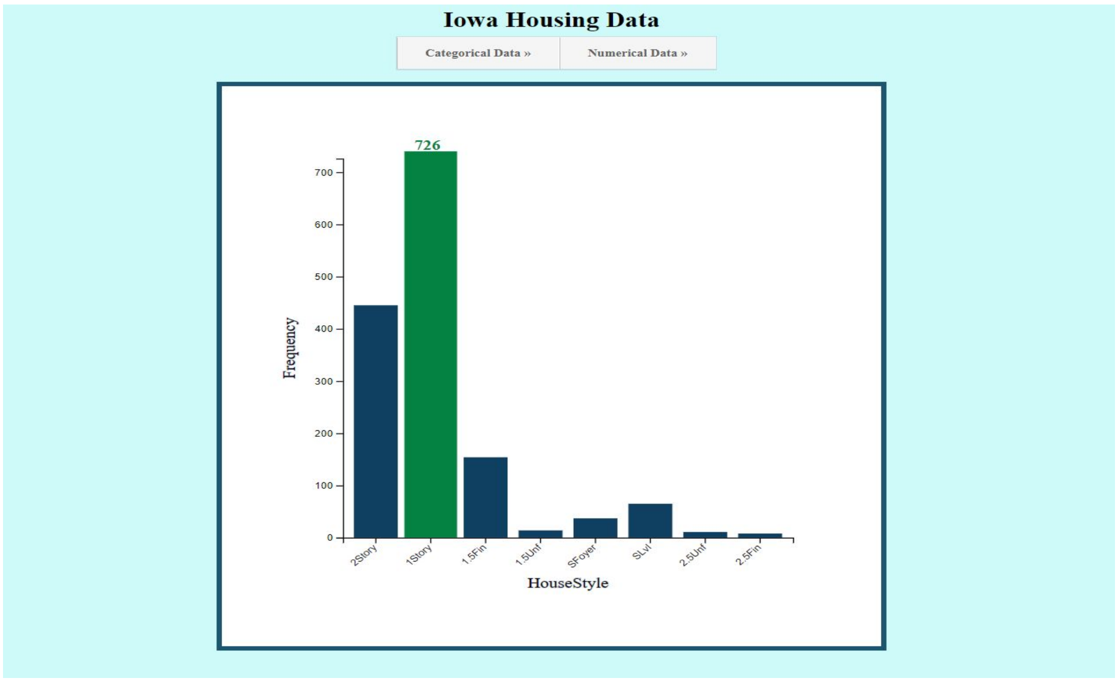
.

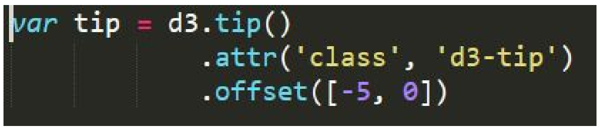
3. Plotting categorical and numerical data: We have two distinct functions *createCategorical()* and *createNumerical()* that handle plotting graphs for categorical and numerical data respectively. We are creating an svg element inside our html <div> and then appending all UI elements to this svg. On choosing another variable, this corresponding function is called again and old contents of svg are erased.



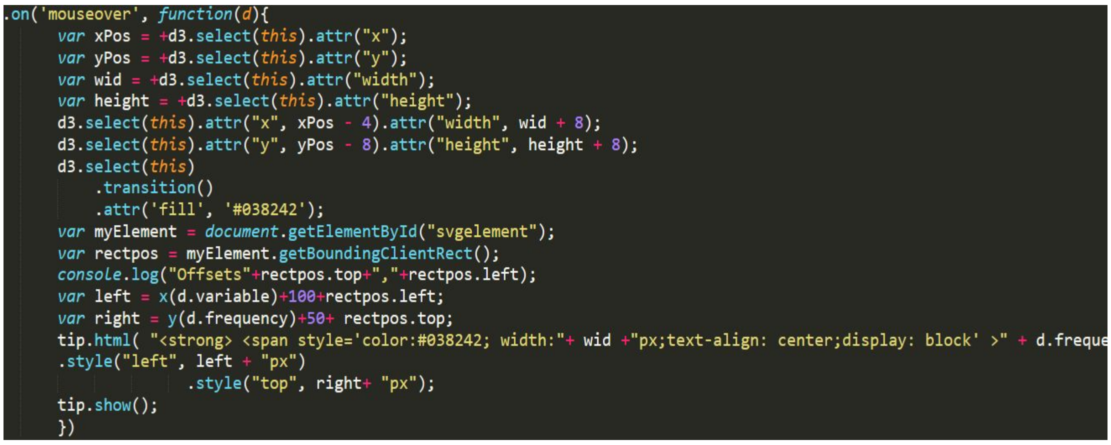
```
var svg = d3.select("#chart").append("svg")
            .attr("width", width + margin.left + margin.right)
            .attr("height", height + margin.top + margin.bottom)
            .attr("id", "svgelement")
            .append("g")
            .attr("transform",
                  "translate(" + margin.left + "," + margin.top + ")");
```

4. Adding tool-tip and resizing selected bar in the graph: The tool-tip has been implemented using d3.tip. I have used the *mouseover* and *mouseout* events to show or hide the tool-tip and to change the bar size and color.

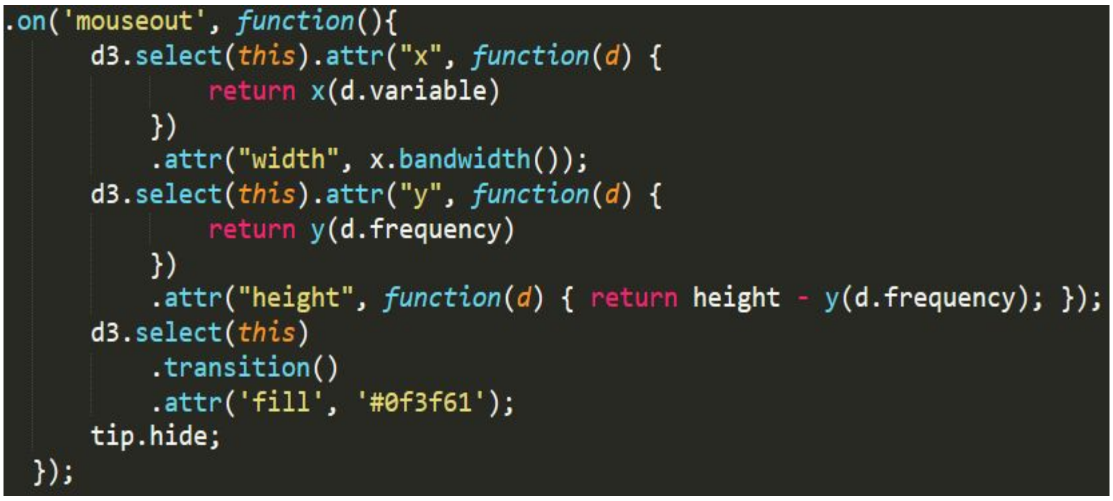




```
var tip = d3.tip()
            .attr('class', 'd3-tip')
            .offset([-5, 0])
```



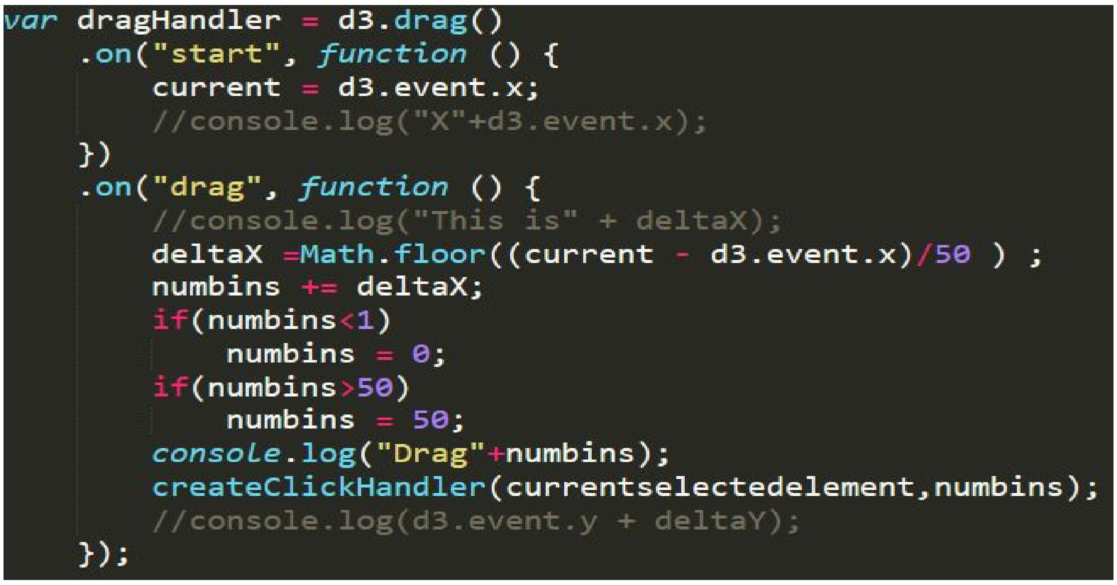
```
.on('mouseover', function(d){
      var xPos = +d3.select(this).attr("x");
      var yPos = +d3.select(this).attr("y");
      var wid = +d3.select(this).attr("width");
      var height = +d3.select(this).attr("height");
      d3.select(this).attr("x", xPos - 4).attr("width", wid + 8);
      d3.select(this).attr("y", yPos - 8).attr("height", height + 8);
      d3.select(this)
          .transition()
          .attr('fill', '#038242');
      var myElement = document.getElementById("svgelement");
      var rectpos = myElement.getBoundingClientRect();
      console.log("Offsets"+rectpos.top+","+rectpos.left);
      var left = x(d.variable)+100+rectpos.left;
      var right = y(d.frequency)+50+ rectpos.top;
      tip.html( "<strong> <span style='color:#038242; width:"+ wid +"px;text-align: center;display: block' >" + d.freque
      .style("left", left + "px")
                  .style("top", right+ "px");
      tip.show();
      })
```



```
.on('mouseout', function(){
        d3.select(this).attr("x", function(d) {
                return x(d.variable)
            })
            .attr("width", x.bandwidth());
        d3.select(this).attr("y", function(d) {
                return y(d.frequency)
            })
            .attr("height", function(d) { return height - y(d.frequency); });
        d3.select(this)
            .transition()
            .attr('fill', '#0f3f61');
        tip.hide;
  });
```

5. <u>Mouse click and drag to change bin size:</u> This functionality has been implemented using d3.drag(). We are using a factor of change of mouse position to adjust the bin

sizes. Initial bin size is set to default value of 10. Bin size is changed on clicking mouse hold and dragging it left or right.



```
var dragHandler = d3.drag()
    .on("start", function () {
        current = d3.event.x;
        //console.log("X"+d3.event.x);
    })
    .on("drag", function () {
        //console.log("This is" + deltaX);
        deltaX =Math.floor((current - d3.event.x)/50 ) ;
        numbins += deltaX;
        if(numbins<1)
            numbins = 0;
        if(numbins>50)
            numbins = 50;
        console.log("Drag"+numbins);
        createClickHandler(currentselectedelement,numbins);
        //console.log(d3.event.y + deltaY);
    });
```

Code set up and execution:

The code will require a web server like XAMPP to execute correctly, since file loading cannot be implemented using just the client side. Please follow the following steps to set up and run the code:

1. Unzip the folder and copy the content folder in //XAMPP//htdocs.
2. Go to http://localhost/pratik/

Please refer to the video demonstration of the project here.