Report for HW-1

**Introduction**

This is a formal report for the Homework 1 set in the Computer Visualization (CSE-564) course. This document describes the requirements given and the approach taken to fulfil the requirements. The purpose of this Homework was to familiarize with the D3 and other JavaScript frameworks. In the sections below, we describe the requirements one-by-one and the approach taken for that sub problem. First, we talk about the project structure:

**Project Structure:**

The following is the directory structure of the files in the homework-

**Hw1/**

**-js/**

Contains all the javascript files. The file is all the rendering functions is render.js

**-css/**

Contains all the css files (bootstrap etc). The custom css file is render.css

**-data/**

Contains the csv file for bar/pie chart and json file for force directed graphs

**create\_json\_file.py-** Python script to generate the JSON file from CSV file

**index.html-** Index file

**force.html-** Extra Credit work. Force directed graph.

The goal is to create a D3 based visual interface which is which should be able to perform certain tasks which are explained below.

**Task 0- Choose a Data Set:**

This was an implicit task to choose a data set intelligently and use it to demonstrate the requirements of the Problem Set. For this homework, I have chosen the data for Basket Ball players. The reason behind selecting this data set is straight forward. The data set has interesting properties for each player like name, height, weight, average, homeruns etc. The data does not look complex and a lot of analysis can be done on this e.g. What is the average of the players in a height range, what is the home run count of the players that have weight between a given interval etc.

**Task 1- Pick a Variable and bin it into a fixed range (equi-width) of your choice-**

Binning is a very important concept used when the data set is scattered. The variable chosen for this section is **Weight.** The reason behind choosing this parameter is purely for the aesthetics of the interface. The graph that comes out of this parameter is more uniformly distributed as expected (More people of average weight).

The top level function to draw a bar chart is render() which generates a bar chart given a parameter as input. Inside the render function we first process the data to generate the data in the same format as expected by the histogram function. We also create a scale function which maps the domain of the data to a specified range. A nice feature of this interface is the color of the bars. D3 provides various color schemes (Category10, 20, 20b etc) for this purpose. We define a ordinal scale which gives us a unique color for each rectangle. The fact that the number of bins will not (usually) exceed 20 (the number of unique colors) here. D3 provides a function (d3.histogram) to generate the histogram of a given dataset. The histogram function takes **thresholds** as a parameter. This value (if given as an integer) denotes the number of bins we want this data to be divided into. We have used this property of d3 histograms to make the interface more flexible and implement the 7th Task. Once the bars are created, we append the rect objects to them with the necessary properties.

**Code Snippet-**

****



**Task 2: Create a Bar chart of the variable picked-**

To create a bar chart of the data, most of the steps are already done. We already have the data binned. We just need to display it on the screen. We use rects for that purpose. The coordinates for the rectangle are automatically calculated by the histogram function. This is where we set all the properties for each bar separately at once (x, y, fill, id etc).

**Code Snippet-**



**Task 3: Using a menu, allow users to select a variable and update chart**

In this task, we must provide the user a menu of all the parameters and upon selection of a menu option, update the chart accordingly. Bootstrap.js framework has been used to create the menu and the dropdowns. To populate the parameter names, we process the first line of the data to get the fields. We drop “name and handedness” from the data set because they do not add much information to this graph (Since handedness can take only two values and binning a data based on name doesn’t make sense). Once we have the fields, we update the menu with appropriate options dynamically using javascript. We also attach an event handler (to be called on mouse clicks) named onclickListenner which calls the render function with the selected parameter. To update the SVG, we first clear it by setting the container div to empty and then drawing the chart again with the update parameter.

**Code Snippets-**

****

****

**Task 4 and 5: On Mouse over, widen the bar and show the data value-**

In this task, we have to implement the event handlers for mouse move and mouse out events. Since we need to capture the mouse activity only on the bars, we deduce that rect objects will be the best place to add the event handlers. In the mouseover event handler, we increase the width and the hight of the bar so that it stands out, changing the color to red at the same time. On the mouseout event handler, we reduce the rect to its original size and color (retrieved using id and the ordinal scale). For the second part, we use a library- d3.tip. It is an open source library for tooltips. Another option can be to show the text directly on top of the bars (mean of the x0 and x1) but we do not take that approach since we need to calculate the new mean each time we hover and change the dimensions of the rect.

**Code Snippets-**

****

****

**Task 6: On Mouse Click, Transform the bar chart into a pie-chart**

To achieve this task, we write a render\_pie function which is a top-level function to generate a pie chart from a data set. We call this function with the parameter chosen by user using the menu provided (like the bar chart). To capture the mouse click event, we create a button and define a onclick handler which, in turn calls the render\_pie function. This handler is automatically invoked when the button is pressed and the pie chart is generated.

To draw the pie chart, we first bin the data using the same method as used in the var charts. After that, the construction of pie chart has two main parts left. First, we need to create an arc generator function which generates the arc for the pie chart. Second, we need to define a pie generator function which calculates the angles and creates the chart given the data set. After that we append the path (lines) to the arcs and setting other attributes etc. The color of the arcs is randomly generated using the same logic as bar charts.

**Code Snippets-**

****

**Task 7: Modify the bin size on mouse movement.**

The goal of this task is to enable the user to control the bin width of the histogram. To achieve this, we dedicate a global variable in javascript to denote the binsize. We create a button to enable the user to modify the value of the binsize. Event handlers are used to capture the mouse movements inside the button. Coordinates of the mouse relative to the left boundary of the button is interpreted as binwidth. Once the binsize is set, we can call the appropriate render function which will render the chart with the selected bin width.

**Code Snippets-**

****

**Extra Credit: Force Directed Graphs-**

The goal of this task was to create a force directed layout based on d3. To create a force directed graph, the major challenge is to create the dataset in the format expected by d3. Since it is a tree based graph, it would be easier if the data was in an ordered dictionary so that we can skip the overhead of the graph on page load. To create the data, a separate python script (create\_json\_data.py) is used. The script when run, parses the csv file to generate a JSON file (baseball\_data\_force.json). The logic to create the graph has been kept simple consideration the fact the purpose of the homework is to focus on visualization rather than data mining. The logic is to create a node for each player. We add an edge between two nodes when the increasing weight of one corresponds to increased average.

The function to render the force graph is render\_force and is kept in render.js file along with other helper functions. The code simply walks through the JSON file creating nodes and edges per the json object.

**Code Snippets-**

****

**Installation and Setup Instructions-**

Installation and setup of this project includes minimal overhead. Before running this project, we need to verify that a web server is installed and running. Installing and starting a webserver is simple. MacBook comes with Apache webservers which can be started using the command-



Similarly, we can use the python inbuilt http server by going to the top level directory (Hw1) and running the command-



Running this command will start a server on port 8888. We chose port 8888 because on a lot of systems, the firewalls tend to block the port 80.

Once the webserver is running, we can open the browser and type the following URL n the address bar-



The URL should open the webpage containing the charts.

**Libraries/Frameworks Used-**

All the libraries used in the development of this interface are open source. The following is the list-

1. D3
2. Bootstrap (CSS and JS)
3. JQuery
4. D3-tip