# CSE 564 Visualization Lab 1 Report

**Dataset chosen:** FIFA 19 complete player dataset (https://www.kaggle.com/karangadiya/fifa19)

This dataset contains detailed attributes for every player registered in the latest edition of FIFA 19 database. It has 18.2k rows and 89 attributes. A large number of the attributes are numerical. I have chosen 19 numerical attributes for the purpose of this assignment. They are as follows:

- Overall score
- Balance
- Stamina
- Strength
- Heading Accuracy
- Short Passing
- Long Passing
- Dribbling
- Ball Control
- Acceleration
- Sprint Speed
- Agility
- Shot Power
- Aggression
- Jumping
- Vision
- Composure
- Standing Tackle
- Sliding Tackle

**Code structure:**

My code consists of one HTML file (index.html) and five JavaScript files, apart from other CSS files and image files.

**index.html:**

This is the starting point of my code; it is the first file which is displayed when the software is launched. Here, I have included a header, a vertical navigation bar for displaying the attributes, a range slider and a couple of images.

**index.js:**

In this file, I have included functions for all the attributes. These functions further call other JavaScript functions which are responsible for drawing charts. I also read the value from the range slider in order to pass it to the functions which draw the charts. Which function to call is decided based upon a global variable “chartType”.

Also, I read data from the CSV file in index.js. Separate arrays are created for each attribute.

I also handle the operation of the range slider in this file. The onchange listener of the range slider which is written in this file changes the number displayed on the screen and updates the chart correspondingly.

**bar_chart.js:**

This file is where the pie chart is drawn. The data needs to be segregated into bins. I have performed the calculations for forming the bins as follows:

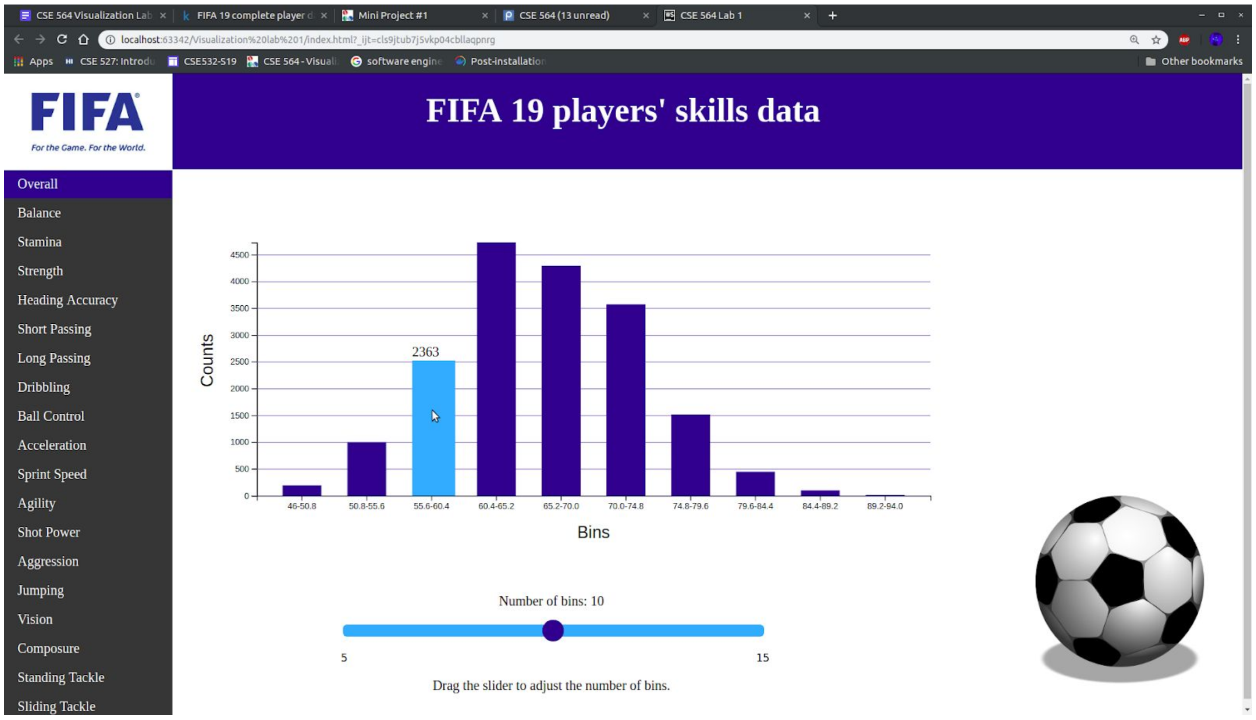
```
function drawBarChart(data, noOfBins) {
   document.getElementById("mySliderContainer").style.visibility = "visible";
   var binSize = (d3.max(data) - d3.min(data))/noOfBins;
   var binValArray = Array.apply(null, Array(noOfBins)).map(Number.prototype.valueOf,0);
   data.forEach(function (d) {
      binValArray[Math.floor((d - d3.min(data))/binSize)]++;
   });

   var min = d3.min(data);
   var binValues = [];
   for(var i = 0; i < noOfBins; i++) {
      var end = (+min + +binSize).toFixed(1);
      binValues.push(min + "-" + end);
      min = end;
    }
    /* rest of the code */
}
```

Further, the bar chart is drawn by binding data and appending elements as required. Transitions and horizontal lines are added to make the bar chart easier to read and to look better. The horizontal lines are added by simply adding another scale with a “-width” inner tick size., so that the tick spans the entire width, thus making a horizontal line.

```
        g.append("g")
        .call(d3.axisLeft(y).tickFormat("").ticks(10).tickSizeInner(-width)).attr("class", "grid");
```

The class grid is used to define the properties of the horizontal lines.
Positioning the mouse pointer over a bar highlights it and displays the value on top. Clicking on any bar transforms the bar chart into a pie chart.

A screenshot of the bar chart is as follows:



**pie_chart.js:**

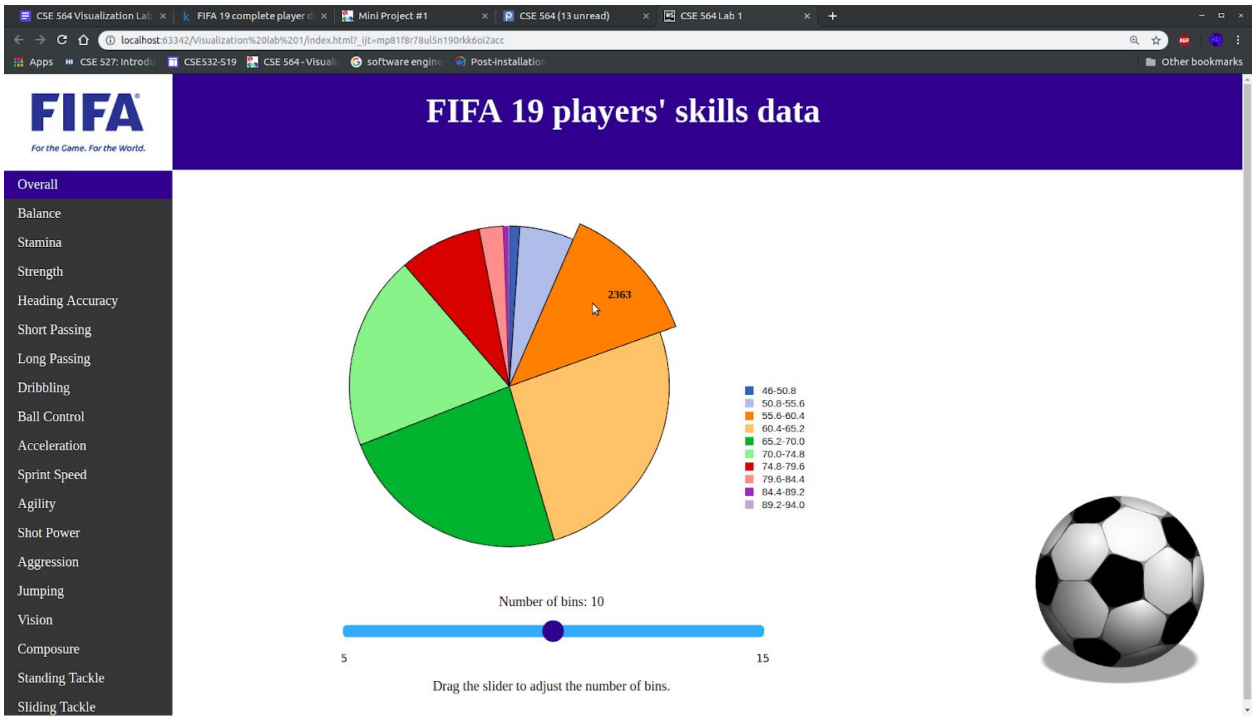
Initially, the data is segregated into bins. Further, the pie layout is used in order to find out the correct angles for the arcs.

```
var pie = d3.pie()
        .sort(null)
        .value(function(d) { return d; });
```

sort(null) ensures that the bins are not sorted according to their size, and displayed on the chart in their original order.

The schemeCategory20 color scale has been used in order to give different color to each bin. A legend has been added to show which bin is represented by which color. Positioning the mouse pointer over an arc increases its size and displays the value. Clicking on any arc transforms the pie chart into a force directed layout.

A screenshot of the pie chart is as follows:



**force_directed_layout.js:**

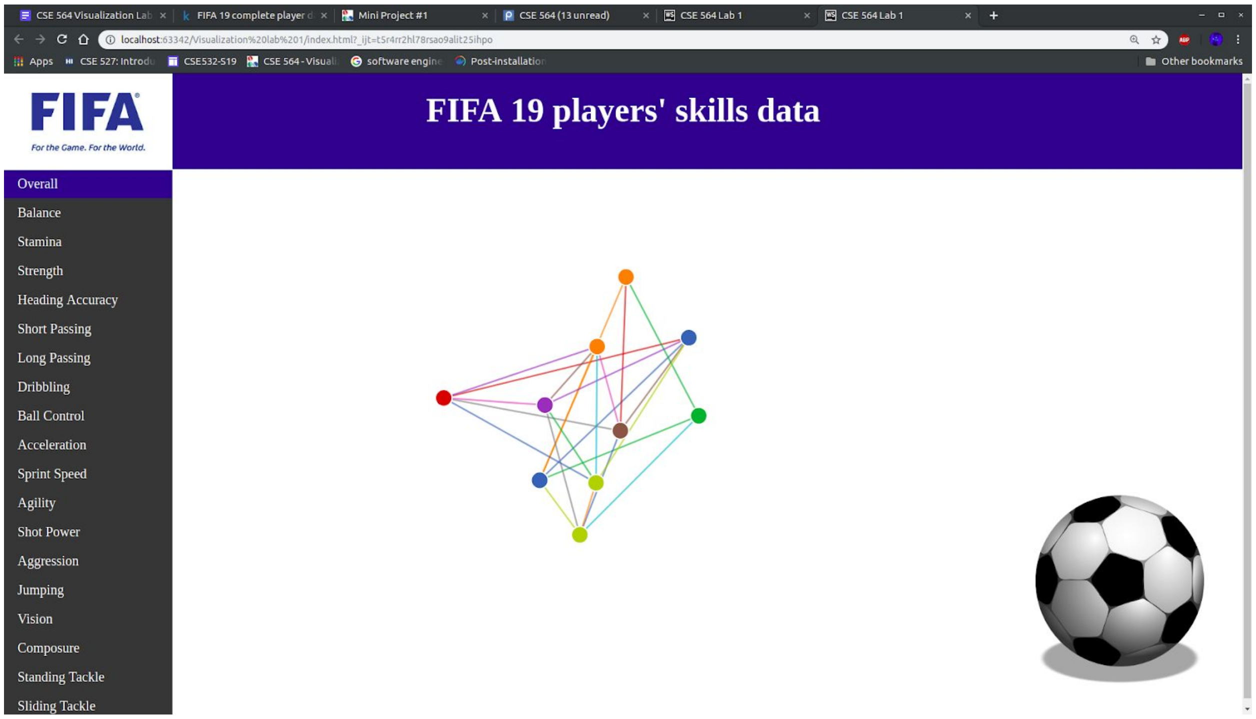
For drawing a force directed layout, I have taken 10 data points with a sampling interval of 500. Since the Euclidean distance needs to be computed for every link, only the first four attributes, i.e. overall, balance, stamina and strength, are considered in order to make the computation faster. Also, I choose and drop links alternately, in order to avoid the chart getting cluttered.

First of all, the nodes and links arrays are set up. Nodes contain the main data while links contain the source and target indices in the nodes array for each link and the Euclidean distance. Then, various d3 functions like d3.forceSimulation(), d3.forceLink(), d3.forceManyBody(), d3.forceCenter() are used to setup the force directed layout.

forceCenter is for setting the center of gravity of the system.
forceManyBody is for making elements attract or repel one another.
forceLink is for creating a fixed distance between connected elements.

The forceCenter has been set to the midpoint of the SVG. The nodes and links are arranged by the force layout, and later, circles are added to the nodes and lines to the links in order to create the chart. Colors have been added using the schemeCategory10 scale.
Clicking on any node transforms the force directed layout into a bar chart.

A screenshot of the force directed layout is as follows.


CSE 564 Visualization Lab
FIFA 19 complete player d
Mini Project #1
CSE 564 (13 unread)
CSE 564 Lab 1
CSE 564 Lab 1
localhost:63342/Visualization%20lab%201/index.html?_ijt=t5r4rr2hl78rsao9alit25ihpo
Apps
CSE 527: Introdu
CSE532-S19
CSE 564 - Visuali
software engine
Post-installation
Other bookmarks
FIFA
For the Game. For the World.
FIFA 19 players' skills data
Overall
Balance
Stamina
Strength
Heading Accuracy
Short Passing
Long Passing
Dribbling
Ball Control
Acceleration
Sprint Speed
Agility
Shot Power
Aggression
Jumping
Vision
Composure
Standing Tackle
Sliding Tackle