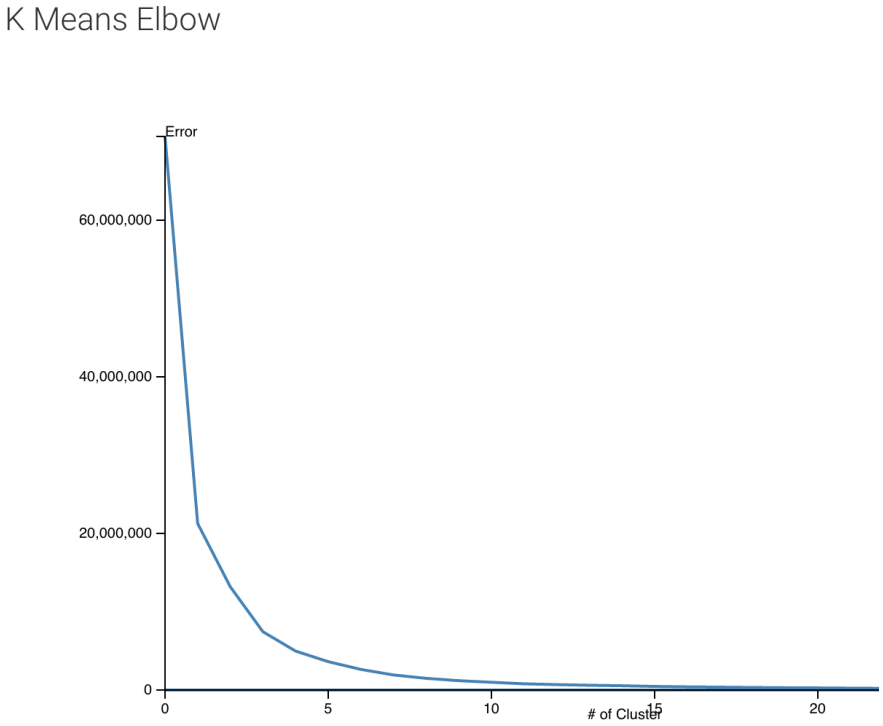
CSE 564 - Data Visualization
Lab 2

Aravind Warrier (110937393)

- **Data Used:**
  Cancer Data
  <https://kaggle.com>
- **Practice the three basic tasks of visual data analytics**
  - **Use data from mini project 1 (or other), begin with $|\mathbf{N}| \geq 500$, $|\mathbf{D}| \geq 10$**
    The data used use a cancer data set from Kaggle.com. I chose 500 data points from the data set. The attributes I used are *radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, concave points_mean, symmetry_mean, fractal_dimension_mean.*
  - **client-server system: python for processing (server), D3 for VIS (client)**
    I used Python with Flask as the backend for hosting a HTTP server and used HTML, CSS, Javascript and d3.js for the VIS.
- **Task1: data clustering and decimation**
  - **Implement random sampling and stratified sampling**
    - **Random Sampling**
      From the 500 data points chosen from the main data set, for random sampling, 286 random data points are chosen. The code for the same is:

      ```
      random.sample (arr, 286)
      ```

      286 is chosen so that both stratified and random sampling give the same number of data points.
    - **Stratified Sampling**
      The 500 data points are clustered first using k-means clustering. There were 3 clusters. The elbow diagram is first plotted to find the right value for k. Following is the elbow diagram:



From each of the three clusters, 50% of the data from each cluster is randomly chosen. Following is the code to randomly chose the data points after doing a k-means clustering:

```
kMeansVar = KMeans(n_clusters=3).fit(new_arr)
for i in label_points:
    data_points_for_i = label_points[i]
    number_of_samples = len(data_points_for_i) * 0.50
    x = random.sample(data_points_for_i, math.ceil(number_of_samples))
    return_list = return_list + x
```

- **The latter includes the need for k-means clustering (optimize k using elbow)**
  K-means clustering was done for Stratified sampling. The diagram is shown above.

```
kMeansVar = [KMeans(n_clusters=k).fit(new_arr) for k in range(1, n)]
centroids = [X.cluster_centers_ for X in kMeansVar]
k_euclid = [cdist(new_arr, cent) for cent in centroids]
dist = [np.min(ke, axis=1) for ke in k_euclid]
wcss = [sum(d ** 2) for d in dist]
```

- **Task 2: dimension reduction (use decimated data)**
  - **Find the intrinsic dimensionality of the data using PCA**
    The intrinsic dimensionality of the data was found out by plotting the eigen values on the correlation matrix found out. The correlation

matrix was built after standardizing the data using the formula:

$$\frac{x - \mu}{\sigma} \tag{1}$$

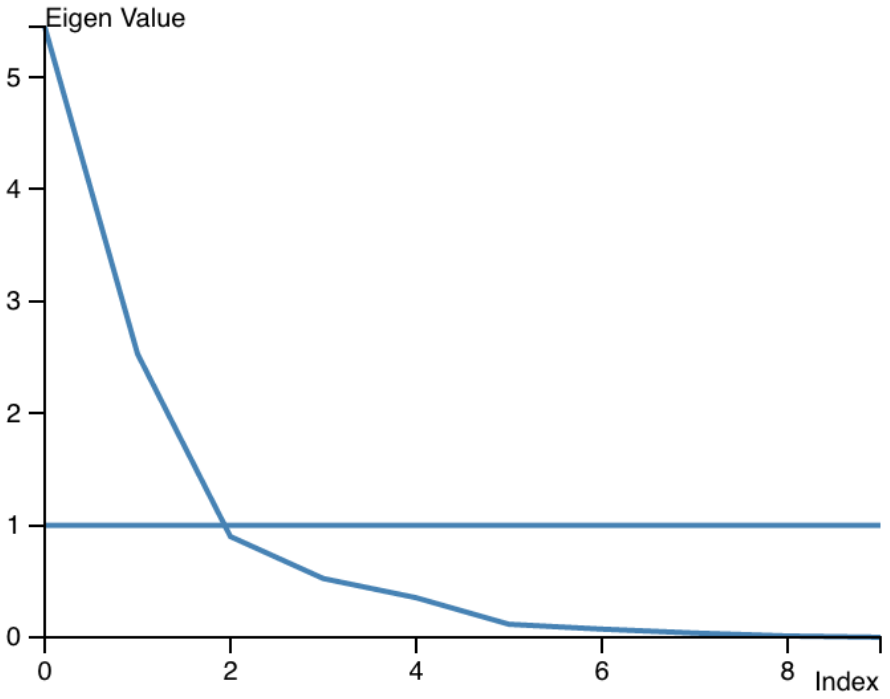
Following is the code snippet find out the eigen values:

```
P, D, Q = np.linalg.svd(arr, full_matrices=False)
```

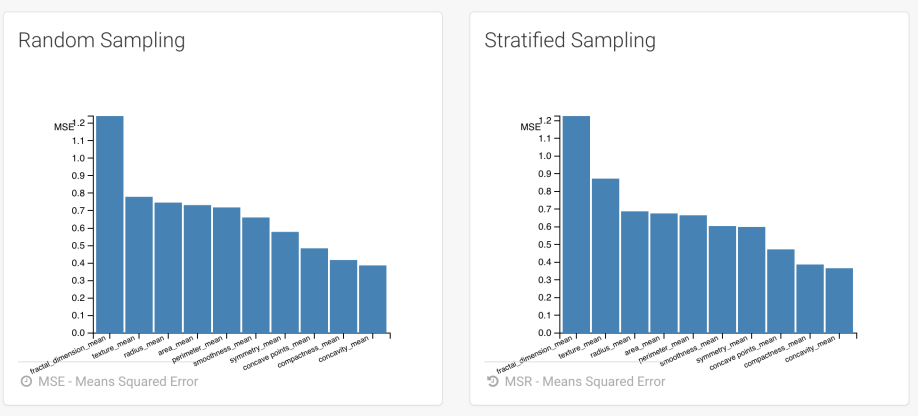
D is the set of eigen values after doing a **Singular Value Decomposition** (SVD)

– **Produce scree plot visualization and mark the intrinsic dimensionality**

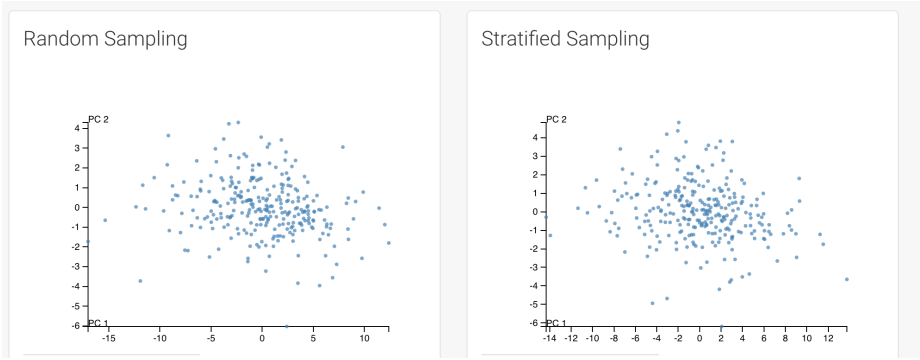
The intrinsic dimensionality graph plotted is shown below:



The scree plot visualization is as shown below:



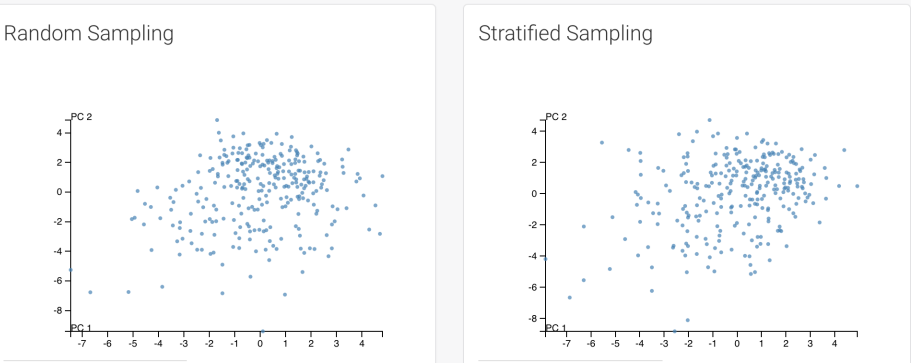
- **Obtain the three attributes with highest PCA loadings**
  The three attributes with the highest PCA loadings are:
  - fractal_dimension_mean
  - texture_mean
  - radius_mean
- **Task 3: visualization (use dimension reduced data)**
  - **Visualize data projected into the top two PCA vectors via 2D scatterplot**
    Following is the scatter plot



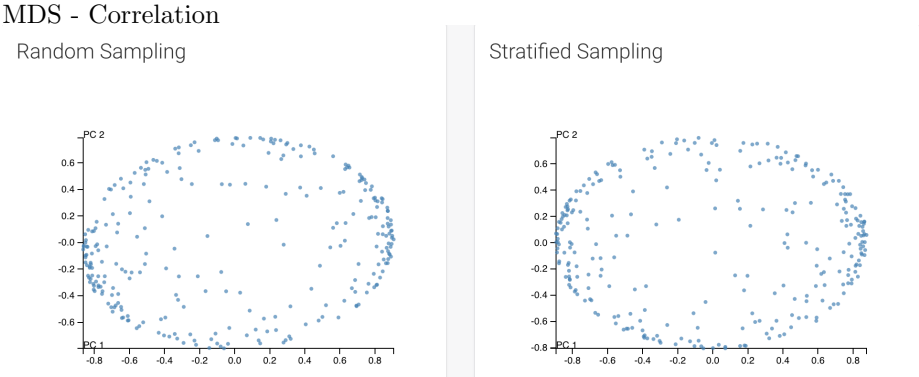
  - **Visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots**

    Following are the visualization made:

    * MDS - Euclidean

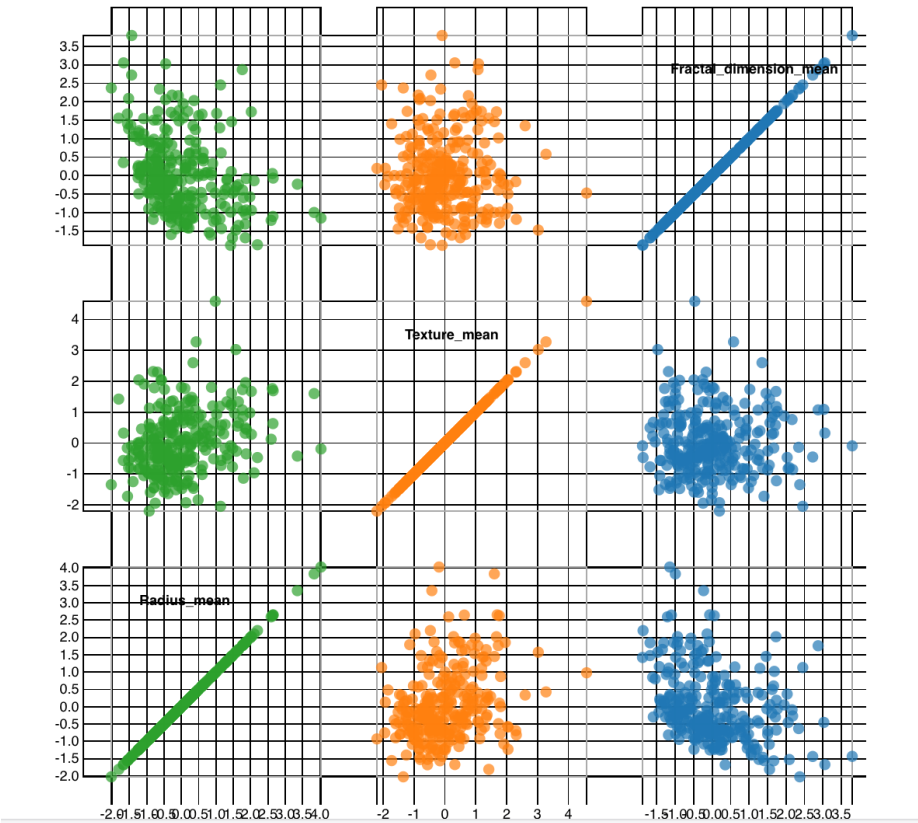


* MDS - Correlation

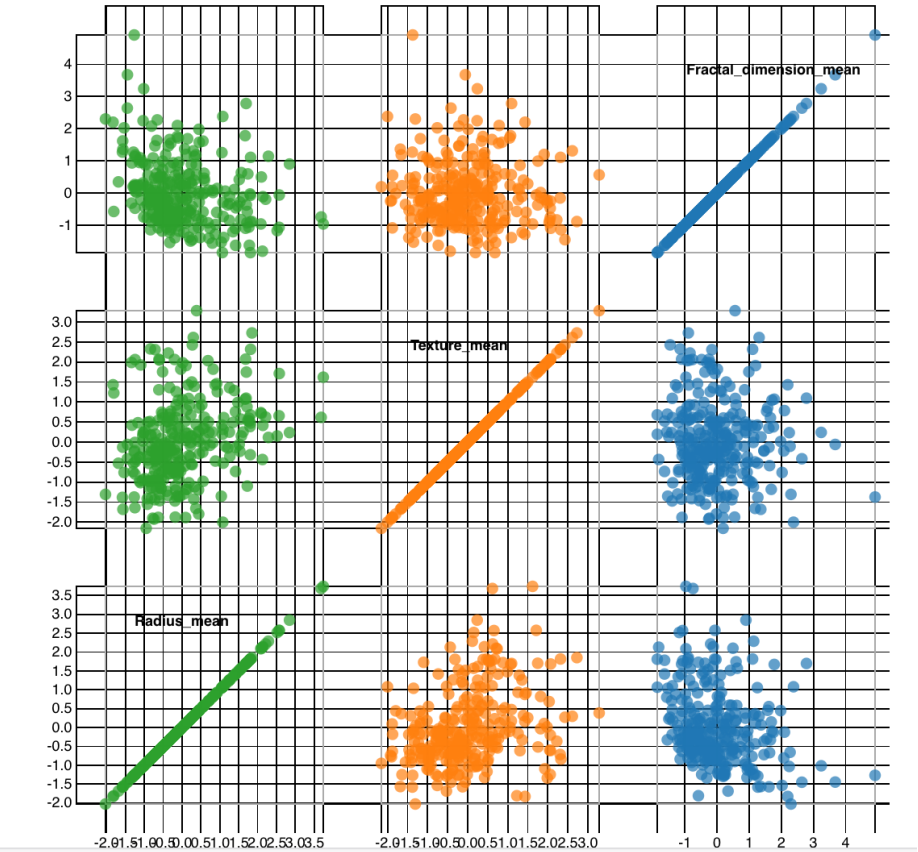


– **Visualize scatterplot matrix of the three highest PCA loaded attributes**

* Matrix Scatter Plot - Random Sample



* Matrix Scatter Plot - Stratified Sample



- **Architecture**
  - **Python HTTP Server**

    The back end uses Python. The HTTPServer is built using Flask. Whenever a new graph/plot is chosen from the UI, a GET request is fired and the Flask Server receives the request and serves the requested page. The Flask server has annotations, using which it recognizes which method to call for each of the GET requests that it receives.
  - **MongoDB**

    I used MongoDB for storing the dataset, when the Flask server receives the first request (GET \ ), the python server fetches the data from the MongoDB, using the command:

```
connection = MongoClient(host, port)
collection = connection[db_name][collection_name]
```

```
return_arr = []
items = collection.find(projection = fields)
```

- **User Interface** The User Interface is built using HTML, Javascript and CSS. I used BootStrap library for presenting nicely. Used D3.js for rendering the line plot, scatter plot, scatter matrix and bar diagram.

- **Code Snippets for visualization**
  - **Bar Chart**

```
    function render_bar(id, data) {
var margin = {
        top: 10,
        right: 15,
        bottom: 60,
        left: 70
    },
    width = 400 - margin.left - margin.right,
    height = 300 - margin.top - margin.bottom;

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

var g = svg.append("g")
    .attr("transform",
    "translate(" + margin.left + "," + margin.top + ")");
var x = d3.scale.ordinal().rangeRoundBands([0, width], .05);
var y = d3.scale.linear().range([height, 0]);

x.domain(data.map(function(d) {
    return d[0];
}));
y.domain([0, d3.max(data, function(d) {
    return d[1];
})]);



var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom")

var yAxis = d3.svg.axis()
```

```
    .scale(y)
    .orient("left")
    .ticks(10);


x.domain(data.map(
    function(d) {
        return d[0];
    }));

y.domain([0, d3.max(data, function(d) {
    return d[1];
})]);


svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis)
    .selectAll("text")
    .style("text-anchor", "end")
    .style("font-size", "8px")
    .attr("dx", "-.8em")
    .attr("dy", "-.55em")
    .attr("transform", "rotate(-25)");

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
    .append("text")
    .attr("y", 3)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text("MSE")
    .attr("transform", "translate(-20,5)")



svg.selectAll("bar")
    .data(data)
    .enter().append("rect")
    .attr("class", "bar")
    .attr("x", function(d) {
        return x(d[0]);
    })
    .attr("width", x.rangeBand())
    .attr("y", function(d) {
        return y(d[1]);
    })
    .attr("height", function(d) {
```

```
            return height - y(d[1]);
        });
}
```

– **Line Plot**

```
        function render_line_plot(id, data, w, h,
        translate_x, x_axis_text, y_axis_text) {

    var margin = {
            top: 10,
            right: 15,
            bottom: 60,
            left: 70
        },
        width = w - margin.left - margin.right,
        height = h - margin.top - margin.bottom;

    var x = d3.scale.linear().range([0, width]);
    var y = d3.scale.linear().range([height, 0]);

    var xAxis = d3.svg.axis().scale(x)
        .orient("bottom").ticks(5);

    var yAxis = d3.svg.axis().scale(y)
        .orient("left").ticks(5);

    var valueline = d3.svg.line()
        .x(function(d) {
            return x(d[0]);
        })
        .y(function(d) {
            return y(d[1]);
        });

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

    // Scale the range of the data

    x.domain(d3.extent(data, function(d) {
        return d[0];
    }));
    y.domain([0, d3.max(data, function(d) {
        return d[1];
```

```
    })]);

    var lineData = [{
        "x": 0,
        "y": y(1)
    }, {
        "x": width,
        "y": y(1)
    }]

    var func = d3.svg.line()
        .x(function(d) {
            return d.x;
        })
        .y(function(d) {
            return d.y;
        })
        .interpolate("linear")

    var lineGraph = svg.append('path')
        .attr("d", func(lineData))
        .attr("stroke", "black")
        .attr("stroke-width", 2)
        .attr("fill", "none")
        .transition()
        .duration(1500)
        .attr('stroke-dashoffset', 0)

    // Add the valueline path.
    svg.append("path")
        .attr("class", "line")
        .attr("d", valueline(data));

    // Add the X Axis
    svg.append("g")
        .attr("class", "x axis")
        .attr("transform", "translate(0," + height + ")")
        .call(xAxis)
        .append("text")
        .text(x_axis_text)
        .attr("transform", "translate(290, 20)")


    // Add the Y Axis
    svg.append("g")
        .attr("class", "y axis")
        .call(yAxis)
        .append("text")
        .text(y_axis_text)
}
```

– **Scatter Matrix**

```
        function render_matrix(id, data, indices, labels) {

    var width = 1060,
        size = 190,
        padding = 40;

    var x = d3.scale.linear()
        .range([padding / 2, size - padding / 2]);

    var y = d3.scale.linear()
        .range([size - padding / 2, padding / 2]);

    var xAxis = d3.svg.axis()
        .scale(x)
        .orient("bottom")
        .ticks(10);

    var yAxis = d3.svg.axis()
        .scale(y)
        .orient("left")
        .ticks(10);

    var color = d3.scale.category10();

    var traits = indices,
    n = traits.length,
    domainByTrait = {}

    traits.forEach(function(trait) {
        domainByTrait[trait] = d3.extent(data,
        function(d) { return d[trait]; });
    });

    xAxis.tickSize(size * n);
    yAxis.tickSize(-size * n);

    var svg = d3.select('#' + id).append("svg")
        .attr("width", size * n + padding)
        .attr("height", size * n + padding)
        .append("g")
        .attr("transform", "translate(" + padding + "," +
        padding / 2 + ")");

    svg.selectAll(".x.axis")
        .data(traits)
        .enter().append("g")
        .attr("class", "x axis")
        .attr("transform", function(d, i) {
```

```
            return "translate(" + (n - i - 1) * size + ",0)";
        })
        .each(function(d) {
            x.domain(domainByTrait[d]);
            d3.select(this).call(xAxis);
        });

    svg.selectAll(".y.axis")
        .data(traits)
        .enter().append("g")
        .attr("class", "y axis")
        .attr("transform", function(d, i) {
            return "translate(0," + i * size + ")";
        })
        .each(function(d) {
            y.domain(domainByTrait[d]);
            d3.select(this).call(yAxis);
        });

    var cell = svg.selectAll(".cell")
        .data(cross(traits, traits))
        .enter().append("g")
        .attr("class", "cell")
        .attr("transform", function(d) {
            return "translate(" + (n - d.i - 1) * size
            + "," + d.j * size + ")";
        })
        .each(plot);

    // Titles for the diagonal.
    cell.filter(function(d) {
            return d.i === d.j;
        }).append("text")
        .attr("x", padding)
        .attr("y", padding)
        .attr("dy", ".71em")
        .text(function(d) {
            return labels[d.x];
        });

    function plot(p) {
        var cell = d3.select(this);

        x.domain(domainByTrait[p.x]);
        y.domain(domainByTrait[p.y]);

        cell.append("rect")
            .attr("class", "frame")
            .attr("x", padding / 2)
            .attr("y", padding / 2)
```

```
            .attr("width", size - padding)
            .attr("height", size - padding);

        cell.selectAll("circle")
            .data(data)
            .enter().append("circle")
            .attr("cx", x(0))
            .attr("cy", y(0))
            .transition()
            .duration(2000)
            .attr("cx", function(d) {
                return x(d[p.x]);
            })
            .attr("cy", function(d) {
                return y(d[p.y]);
            })
            .attr("r", 4)
            .style("fill", function(d) {
                return color(p.x);
            });

    }

    function cross(a, b) {
        var c = [],
            n = a.length,
            m = b.length,
            i, j;
        for (i = -1; ++i < n;)
            for (j = -1; ++j < m;) c.push({
                x: a[i],
                i: i,
                y: b[j],
                j: j
            });
        return c;
    }
}
```

– **Scatter Plot**

```
        function scatter_plot(id, data) {
    var margin = {
            top: 20,
            right: 15,
            bottom: 60,
            left: 60
        },
        width = 400 - margin.left - margin.right,
        height = 300 - margin.top - margin.bottom;
```

```
var x = d3.scale.linear()
    .domain([d3.min(data, function(d) {
        return d[0];
    }), d3.max(data, function(d) {
        return d[0];
    })])
    .range([0, width]);

var y = d3.scale.linear()
    .domain([d3.min(data, function(d) {
        return d[1];
    }), d3.max(data, function(d) {
        return d[1];
    })])
    .range([height, 0]);

var chart = d3.select('#' + id)
    .append('svg:svg')
    .attr('width', width + margin.right + margin.left)
    .attr('height', height + margin.top + margin.bottom)
    .attr('class', 'chart')


var main = chart.append('g')
    .attr('transform', 'translate(' +
    margin.left + ',' + margin.top + ')')
    .attr('width', width)
    .attr('height', height)
    .attr('class', 'main')

// draw the x axis
var xAxis = d3.svg.axis()
    .scale(x)
    .orient('bottom');

main.append('g')
    .attr('transform', 'translate(0,' + height + ')')
    .attr('class', 'main axis date')
    .call(xAxis)
    .append("text")
    .text("PC 1");

// draw the y axis
var yAxis = d3.svg.axis()
    .scale(y)
    .orient('left');

main.append('g')
    .attr('transform', 'translate(0,0)')
    .attr('class', 'main axis date')
```

```
            .call(yAxis)
            .append("text")
            .text("PC 2");

        var g = main.append("svg:g");

        g.selectAll("scatter-dots")
            .data(data)
            .enter().append("svg:circle")
            .attr("cx", function(d, i) {
                return width;
            })
            .attr("cy", function(d) {
                return height;
            })
            .transition()
            .duration(2000)
            .attr("cx", function(d, i) {
                return x(d[0]);
            })
            .attr("cy", function(d) {
                return y(d[1]);
            })
            .attr("r", 2);
    }
```

- **Server Code - Python**
  - **Standardizing Data**

```
    def normalize_data(data):
    data_arr = []

    for datum in data:
        arr = []
        for field in fields:
            if field is not '_id':
                arr.append(datum[field])
        data_arr.append(arr)

    return preprocessing.scale(np.array(data_arr))
```

  - **PCA Calculation**

```
    def calculate_pca(arr, num_of_pc = None):
    """
    This method does the PCA and returns the PC's that have
    eigen values more than 1.
    :param arr:
    :return:
    """
```

```
if num_of_pc is None:

    P, D, Q = np.linalg.svd(arr, full_matrices=False)
    pc_components_len = len([index for index, value in enumerate(D)
    if value > 1])

    pca = PCA(n_components=pc_components_len)
    pca_result = pca.fit_transform(arr)

    squared_pca_result = [square_func(value)
    for index, value in enumerate(pca_result)]
    sum_squared_pca_result = [sum(value)
    for index, value in enumerate(squared_pca_result)]

    field_rms_pca_result = dict()
    index = 0

    for field in fields:
        if field is not '_id':
            field_rms_pca_result[field] = (
            sum_squared_pca_result[index]/pc_components_len)
            ** 0.5
            index += 1

    field_rms_pca_result = sorted(
    field_rms_pca_result.items(), key=lambda x: x[1] * -1)

    return field_rms_pca_result, pca_result, D
```

- **MDS - Correlation and Euclidean**

```
def do_mds():
    global mds_random_euclidean
    global mds_stratified_euclidean
    global mds_random_correlation
    global mds_stratified_correlation
    global normalized_stratified_sample
    global normalized_random_sample


    mds_random_euclidean =
        calculate_mds(normalized_random_sample, 'euclidean')
    mds_stratified_euclidean =
        calculate_mds(normalized_stratified_sample, 'euclidean')

    mds_random_correlation =
        calculate_mds(normalized_random_sample, 'correlation')
    mds_stratified_correlation =
        calculate_mds(normalized_stratified_sample, 'correlation')
```

- **Source Code:** https://github.com/warrieraravind/BarChart_PieChart

- **References:**
  - https://bl.ocks.org/
  - https://www.youtube.com/channel/UC18PYast-g9zjsavyv08oMg