VISUALIZATION LAB 2 REPORT

<u>Technical details and tools used</u>

The project is done using Backend as Python Flask server. The JS queue library is used for accesing the backend APIS.The front-end is designed using HTML,CSS, Javascript. Bootstrap libraries are used for front-end designs.
The HTML code is present in templates folder. The JS and CSS is present in static folder.
server.py and sampling.py handles backend flask server and python code.
Memcache is used to cache data in the server side to pass in subsequent calls.

<u>Task1: data clustering and decimation : implement random sampling and stratified sampling</u>

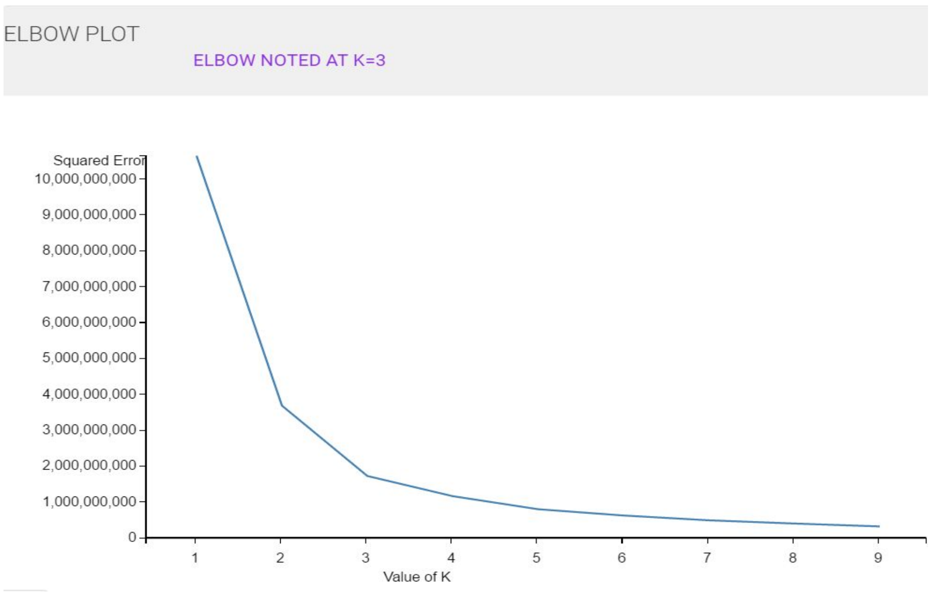
<u>RANDOM SAMPLING</u>

Random sampling is done using the random library, which samples a fraction of data and returns the random results. The dataset file name and the fraction of data can be specified in the configuration file Config.py.

```python
import random
def random_sample(values,num):
    return random.sample(values, int(num))
```

<u>STRATIFIED SAMPLING</u>

Stratified sampling also samples the fraction of data from given dataset, but it differs in such a way that it samples according to the size of bins/clusters. These clusters are formed by using K-Means clustering. The size of the K-means cluster is found by using Elbow Method.
The K-Means elbow is found by plotting the Mean Squared Error against the value of K.
The plot to get K-Means elbow is shown below :



Note that the graph is available as Bar chart and scatter plot as well in the UI, you just need to select that option in the page.
In the given dataset, the elbow is observed at K=3. Hence, for stratified sampling, the code is clustered into 3 clusters and it is sampled with respect to the size of each cluster.

This is the code to get values to plot elbow from the Flask server :

```python
def plot_elbow(values):
    n = 10

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

    return wcss
```

Stratified sampling is then performed :

```python
def stratified_sample(values):

    k = 3
    kMeans = KMeans(n_clusters=k).fit(values)
    classified_values = {}
    for i in range(0,k):
        classified_values[i] = []

    for i,val in enumerate(values):
        classified_values[kMeans.labels_[i]].append(values[i])

    sampled_values = []
    for i in range(0,k):
         sampled_values.extend(random_sample(classified_values[i],
           constants.sample_fraction*len(classified_values[i])))
    return sampled_values
```

The D3 code for plotting the line chart with the values will be explained in a while.

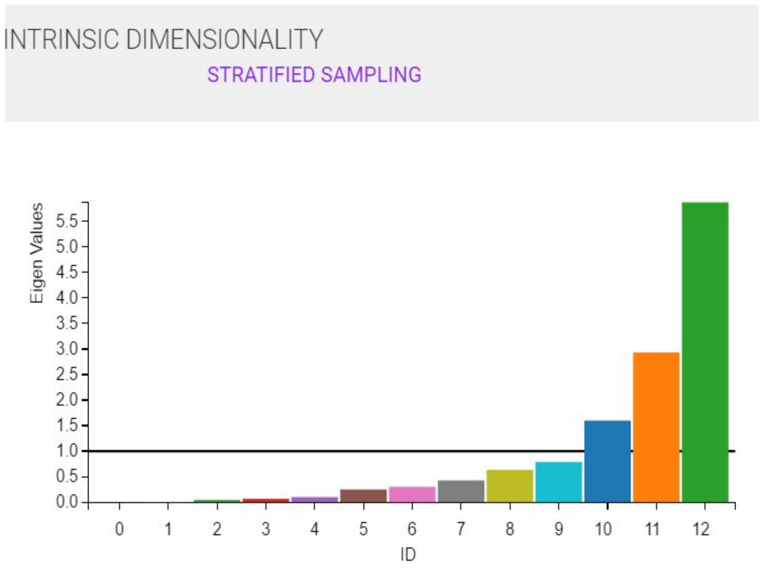
Task 2: dimension reduction (use decimated data)

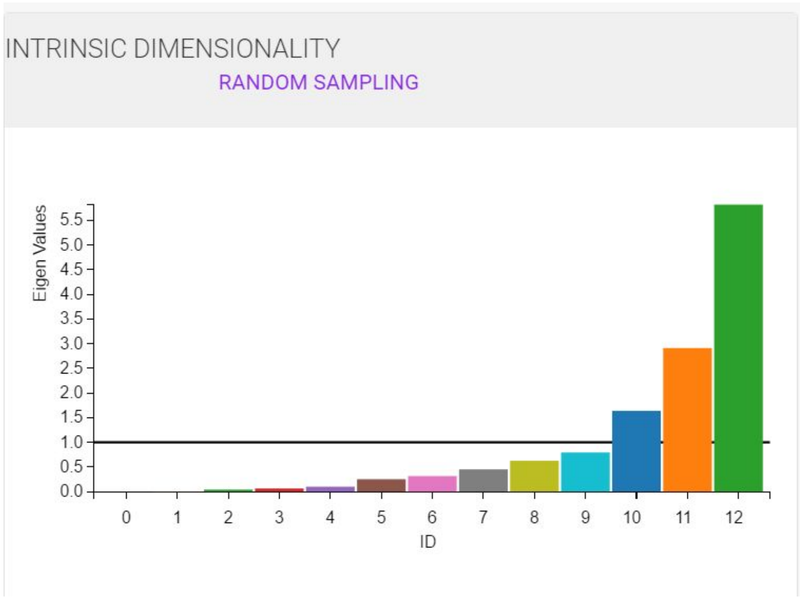
- **find the intrinsic dimensionality of the data using PCA**

PCA (Principal Component Analysis) is done on the data, to get the components which have highest variance in the dataset. Intrinsic dimensionality can be found by plotting the variables Against their eigen values, and taking the variables with eigen value greater than 1.
The eigen values can be obtained as shown :

```
def get_eigen_values():
    random_samples,stratified_samples = get_samples()

    dict = {}

    random_correlation_matrix =  np.corrcoef(random_samples,rowvar=False)
    random_eigen_vals,random_eigen_vector =
                 np.linalg.eigh(random_correlation_matrix)

    stratified_correlation_matrix =
             np.corrcoef(stratified_samples,rowvar=False)
    stratified_eigen_vals,stratified_eigen_vector =
             np.linalg.eigh(stratified_correlation_matrix)
```

This is plotted for both random and stratified samples :





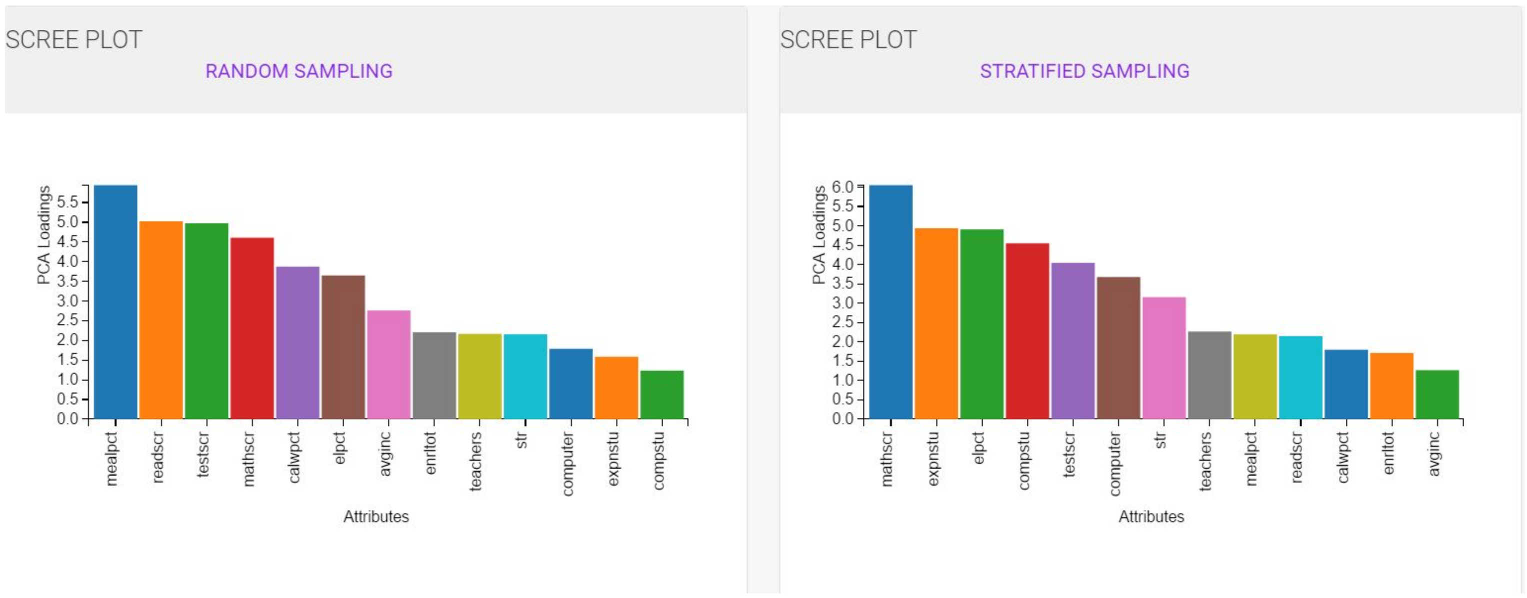
We can see that intrinsic dimension is 3 for the dataset in both random and stratified samplings.

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

The scree plot is plotted against the variables and sum of squared loadings. The PCA components are found using the PCA function in Python sklearn.

```
from sklearn.decomposition import  PCA

   def get_scree_data():
    random_samples,stratified_samples = get_samples()
    random_correlation_matrix =  np.corrcoef(random_samples,rowvar=False)
    stratified_correlation_matrix =
np.corrcoef(stratified_samples,rowvar=False)

    pca =  PCA(n_components = 3)

    random_pca_values = pca.fit_transform(random_correlation_matrix)
    stratified_pca_values =
pca.fit_transform(stratified_correlation_matrix)

    random_scree_values = np.square(random_pca_values)
    stratified_scree_values = np.square(stratified_pca_values)
    random_scree_values = np.sum(random_scree_values, axis=1)
    stratified_scree_values = np.sum(stratified_scree_values, axis=1)
```

The scree plot obtained is shown below for random and stratified samples:



- **<u>obtain the three attributes with highest PCA loadings</u>**

The attributes with highest PCA loadings are obtained from the plot above,
For random samples, the highest loaded PCA attributes are:
Mealpct
Readscr
Testscr

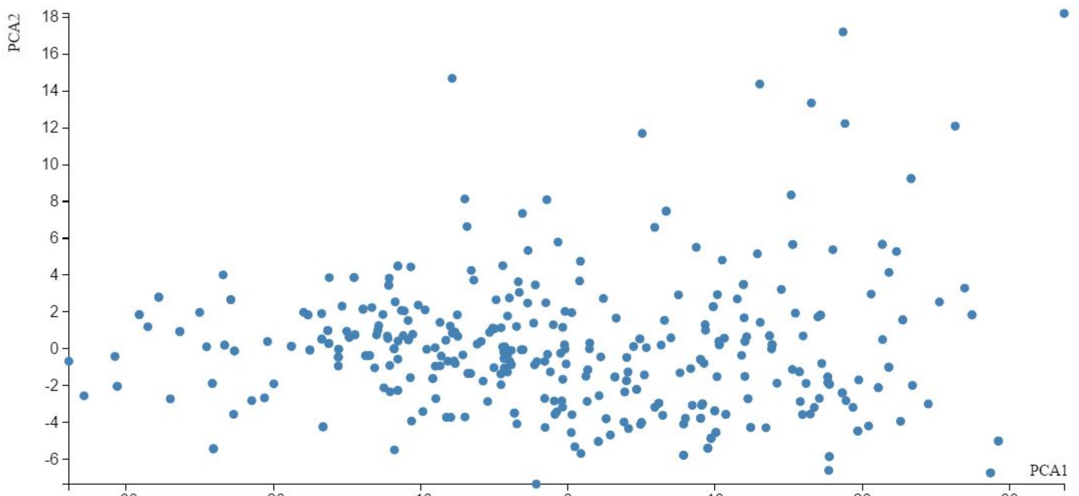
For stratified sampling, highest loaded PCA attributes are:
Mathscr
Expnstu
Elpct

<u>Task 3: visualization</u>

- **<u>visualize data projected into the top two PCA vectors via 2D scatterplot</u>**

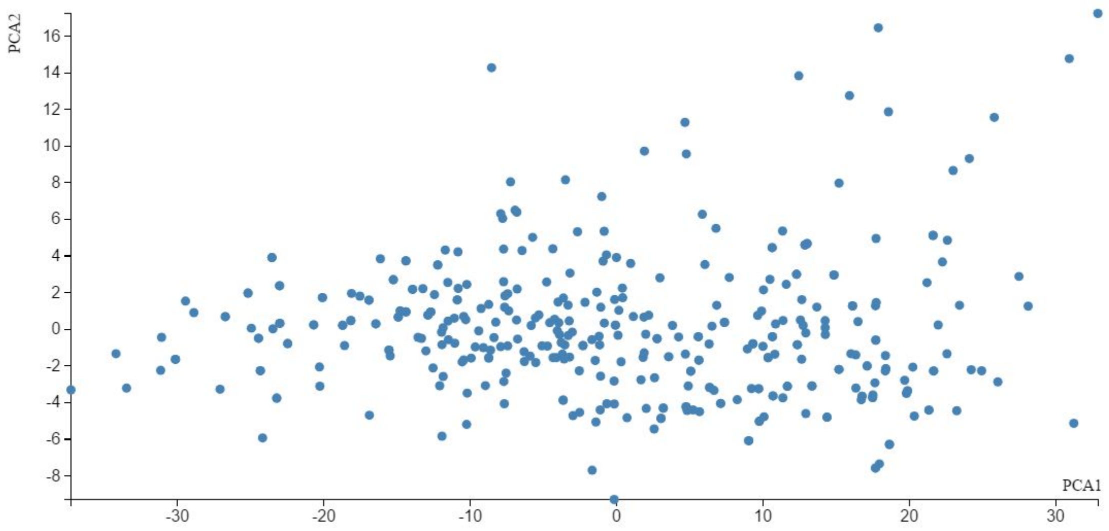
PCA SCATTERPLOT - CLICK TO TOGGLE

RANDOM SAMPLING

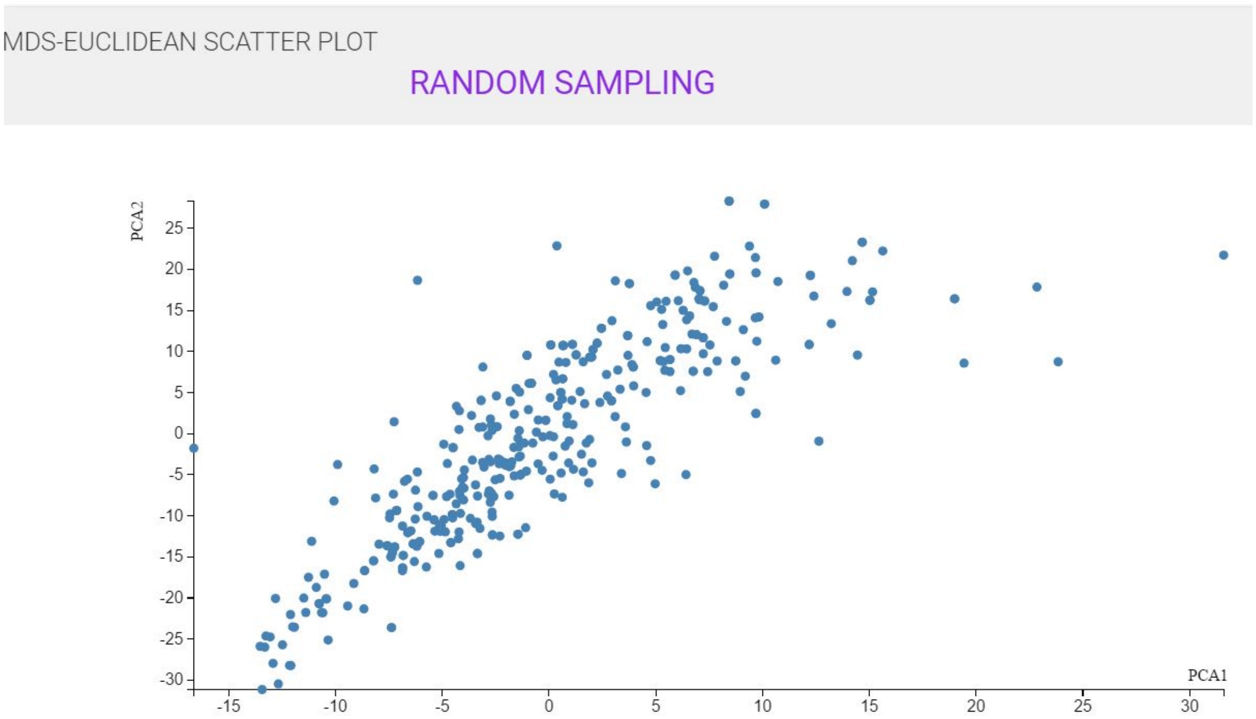


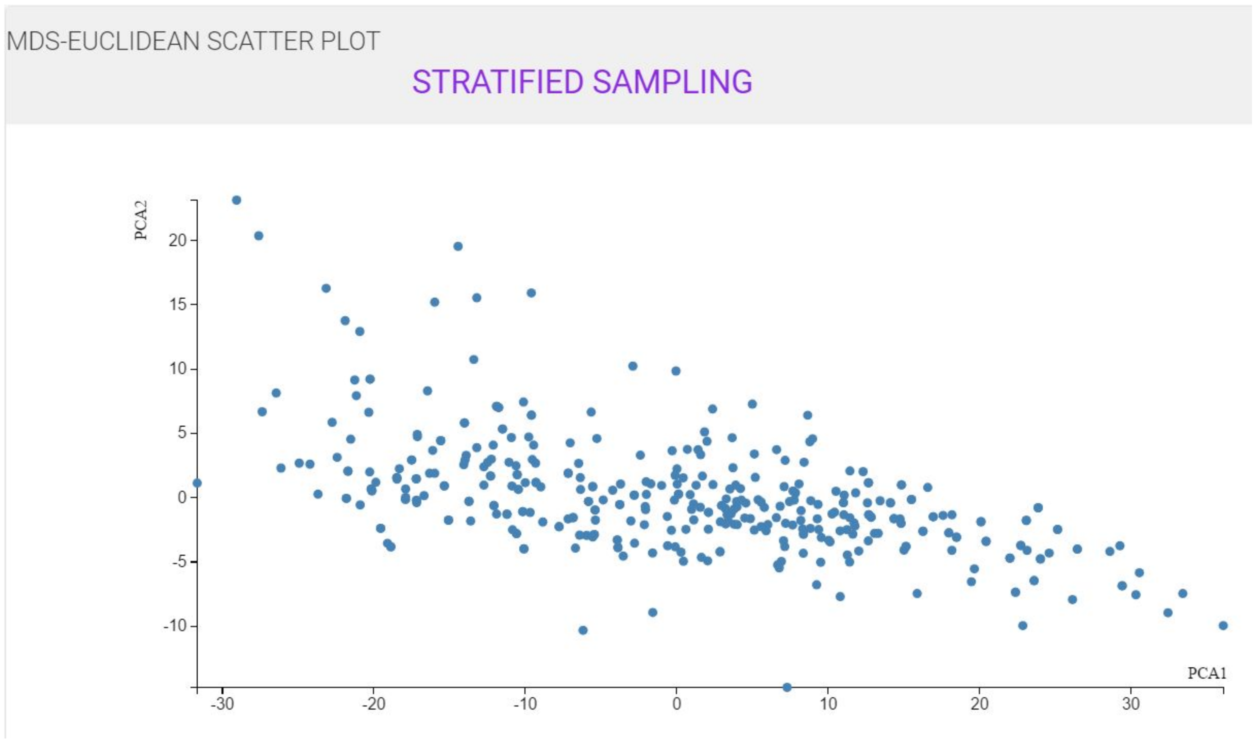
PCA SCATTERPLOT - CLICK TO TOGGLE

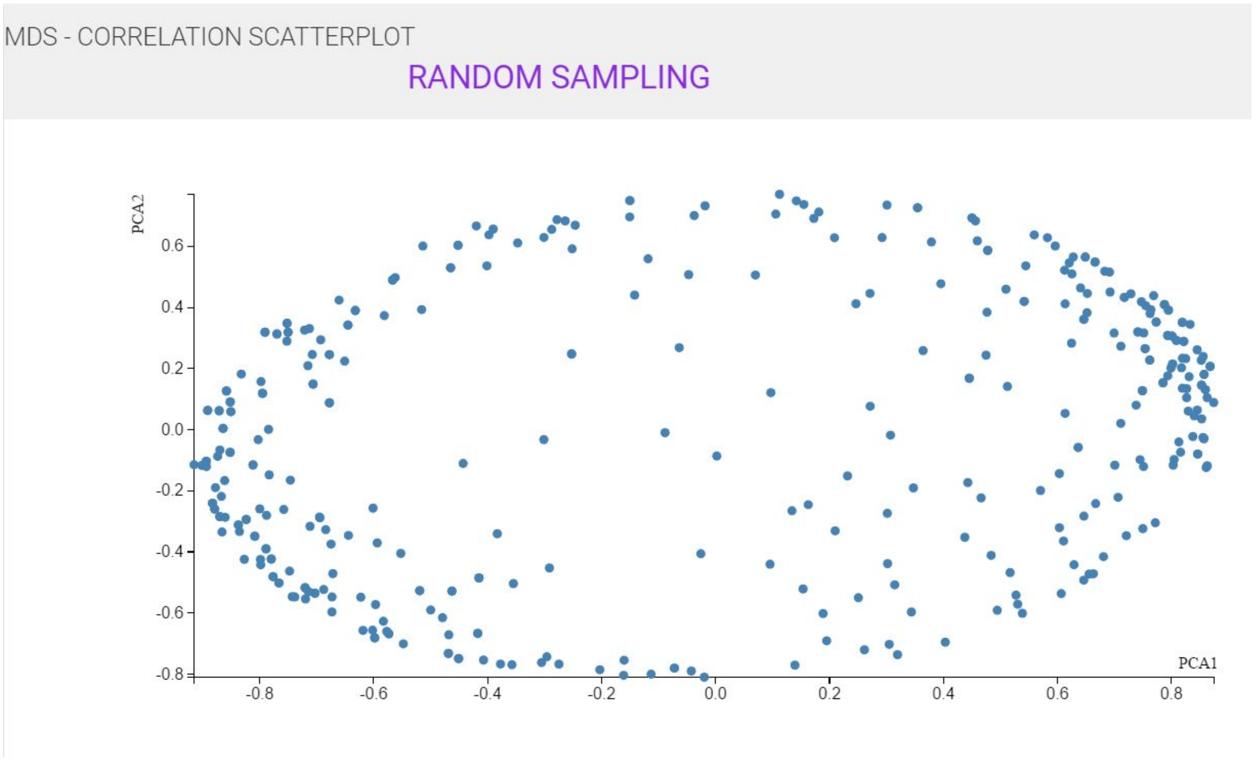
STRATIFIED SAMPLING

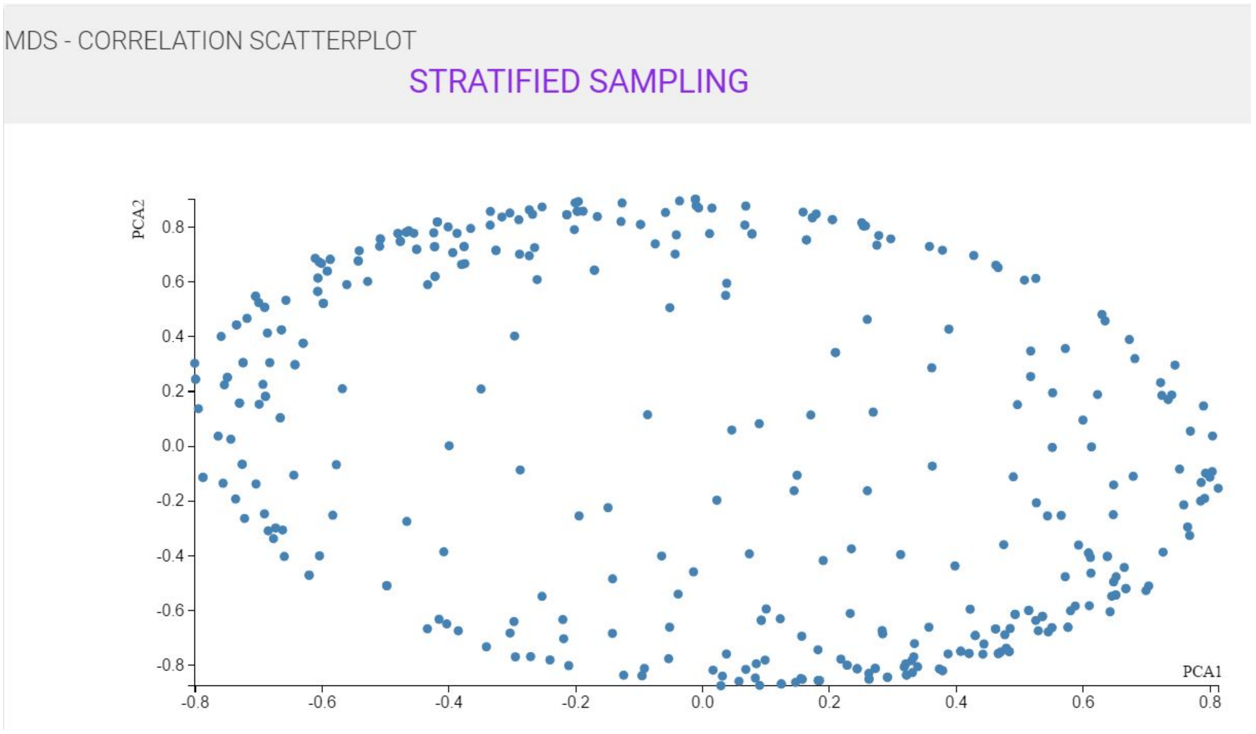


- **visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots**





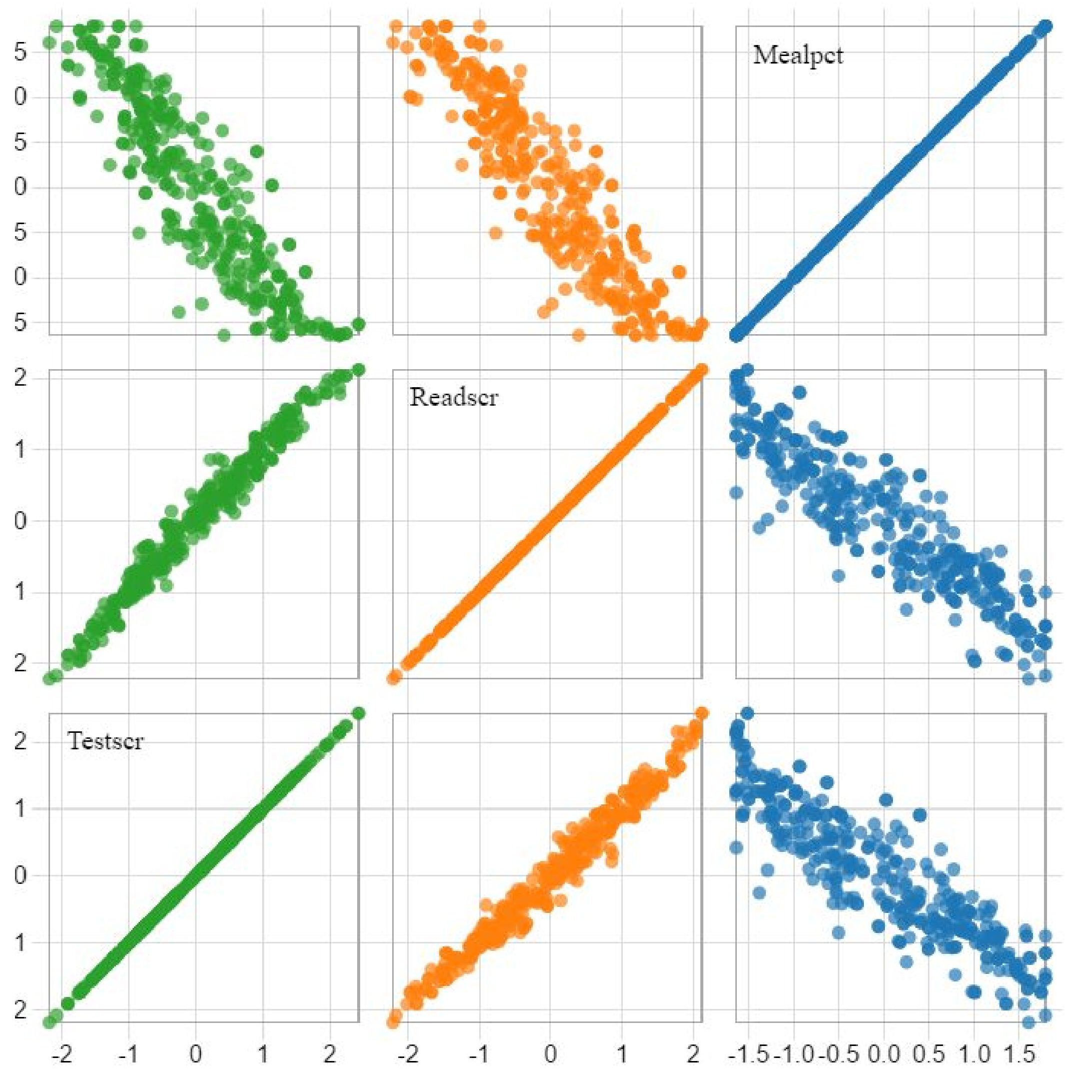


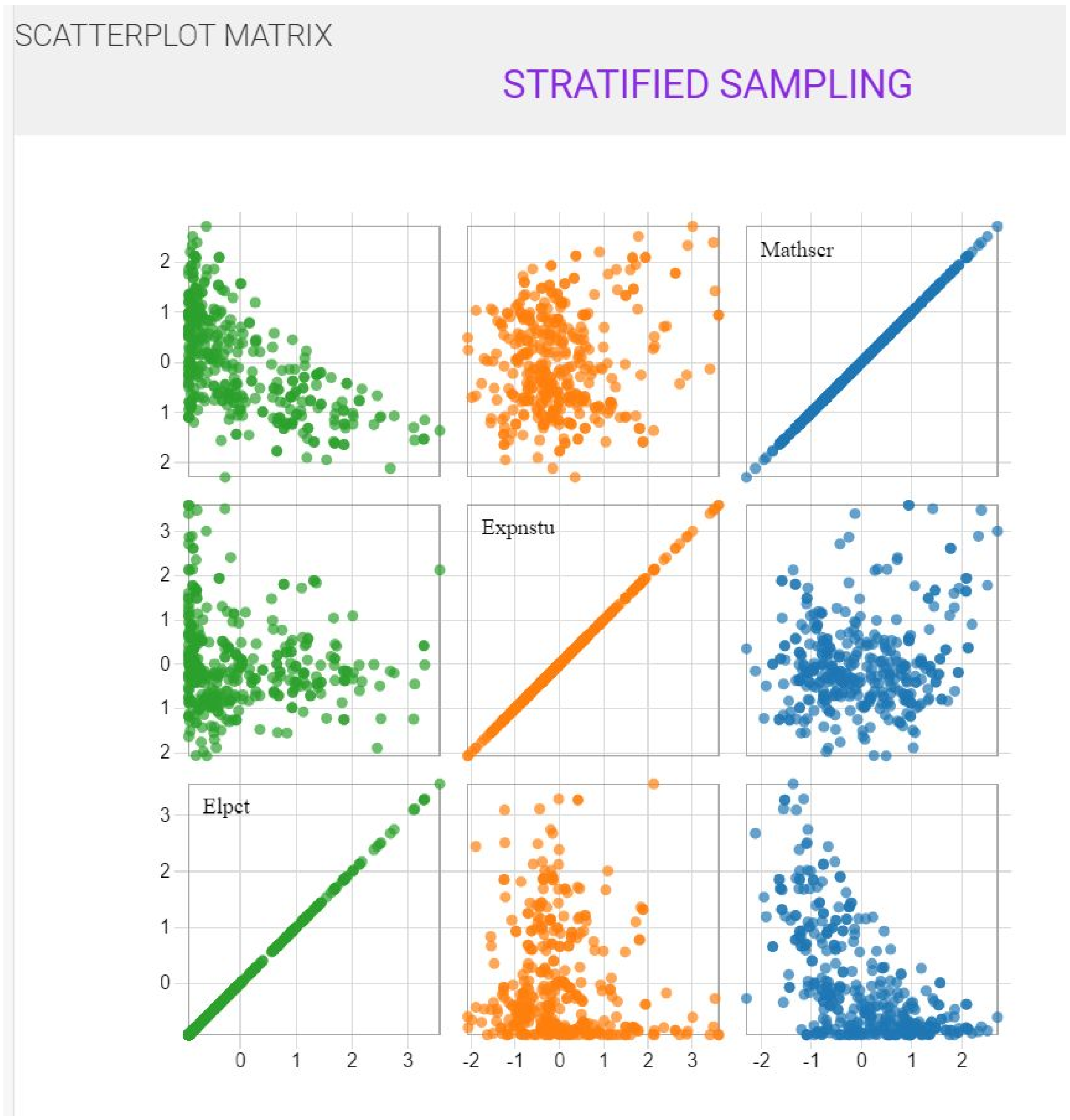


- **<u>Visualize scatterplot matrix of the three highest PCA loaded attributes</u>**

SCATTERPLOT MATRIX

RANDOM SAMPLING





Font- end code Details
All the D3 related JS code is involved in the file static/plot.js

Plotting the scatter plots
Scatter plots are used to plot the PCA components, MDS data as well as in other pages as well. A common code is used for all this, except data is changed using different APIs. The code snippet for scatter plot is shown :

Initialize the offsets and margins, x and y domain and range is set.

```
  var margin = {top: 20, right: 15, bottom: 60, left: 60}
      , width = 960 - margin.left - margin.right
      , height = 500 - 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 ]);
```

The chart is plotted by adding svg, drawing X and Y axes.

```

    var chart = d3.select(id)
      .append('svg:svg')
      .attr('width', width + margin.right + margin.left)
      .attr('height', height + margin.top + margin.bottom+25)
      .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")
```

```
      .style("text-anchor", "end")
      .attr("dx", "-.8em")
      .attr("dy", "-.55em")
      .attr("transform", "translate(900,0)" )
      .text("PCA1")
      .style('font','20px times');

    // 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")
      .attr("transform", "rotate(-90)")
      .attr("y", -55)
      .attr("dy", ".80em")
      .style("text-anchor", "end")
      .text("PCA2")
      .style('font','20px times');
```

The scatter plot is plotted using circles, by providing the co-ordinates, the radius and fill color. A transition is also added to the graph to produce a delayed appearance of the dots from the center of the graph.

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

    g.selectAll("scatter-dots")
      .data(data)
      .enter().append("svg:circle")
          .attr("cx", function (d,i) {
           return x(0);
         } )
          .attr("cy", function (d) {
          return y(0);
          } )
          .attr("r", 4)
          .attr("fill", "steelblue")
                  .transition()
```

```
        .duration(2000)
          .attr("cx", function (d,i) { return x(d[0]); } )
        .attr("cy", function(d) {
            return y(d[1]);
          });
}

```

SCATTERPLOT MATRIX

```
function scatter_matrix(da,side,labels)
{

// ----  Initialize xAxis, yAxis, margins and paddings for svg ----

  var svg = d3.select("#plot_"+side).append("svg")
      .attr("width", size * n + padding+ 100)
      .attr("height", size * n + padding + 30)
    .append("g")
    .attr("transform", "translate(0,0)");

  svg.selectAll(".x.axis")
      .data(index)
    .enter().append("g")
      .attr("class", "x axis")
      .attr("transform", function(d, i) { return "translate(" + (n - i - 1)
* size + ",0)"; })
      .each(function(d) {
        x.domain(domains[d]);
        d3.select(this).call(xAxis);
      });

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

  var cell = svg.selectAll(".cell")
      .data(cross(index,index))
    .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,i) {

        return labels[d.x];
         })
      .style('font','16px times');
```

Plot the cells in scatter plot matrix,

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

    x.domain(domains[p.x]);
    y.domain(domains[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",function(d){return x(0);})
        .attr("cy",function(d){return 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;
}
```

The bar charts are plotted using the same code snippets as Lab 1.
Line charts are plotted using the line() function in svg, a small snippet of main code is shown below:

```
var plotline = d3.svg.line()
    .x(function(d,i) {
      return x(i); })
    .y(function(d) { return y(d); });


var line_graph  = svg.append("g").
    attr("class",'line-graph');

   line_graph.append("path")
        .attr("class", "line")
        .attr("d", plotline(data));
```

<u>REFERENCES</u>
Flask server, queue integration in Python
http://adilmoujahid.com/posts/2015/01/interactive-data-visualization-d3-dc-python-mongodb/

D3 scatter plot matrix details https://bl.ocks.org/mbostock
Bootrstrap templates for UI https://www.creative-tim.com/