Subject: Visualization
Project 2: Data reduction and dimensionality reduction for visualization of data.

NOTE: Check out the "Visualization_A2_Demo.mov" video file to see the actual implementation of project.

This assignment includes visualization of dengue's data from my_dengue.csv in source-code folder. The data have around 500 samples and 14 attributes (dimensions).

It involves the main server code in "server.py" which does all the processing of data and stores the results in flat csv files in source-code/static/data folder provided in the submission.

To test the code, please run the server.py which will create a local server at localhost:5000 and then test the application by visiting the url

http://localhost:5000/

Task 1: Data clustering and decimation

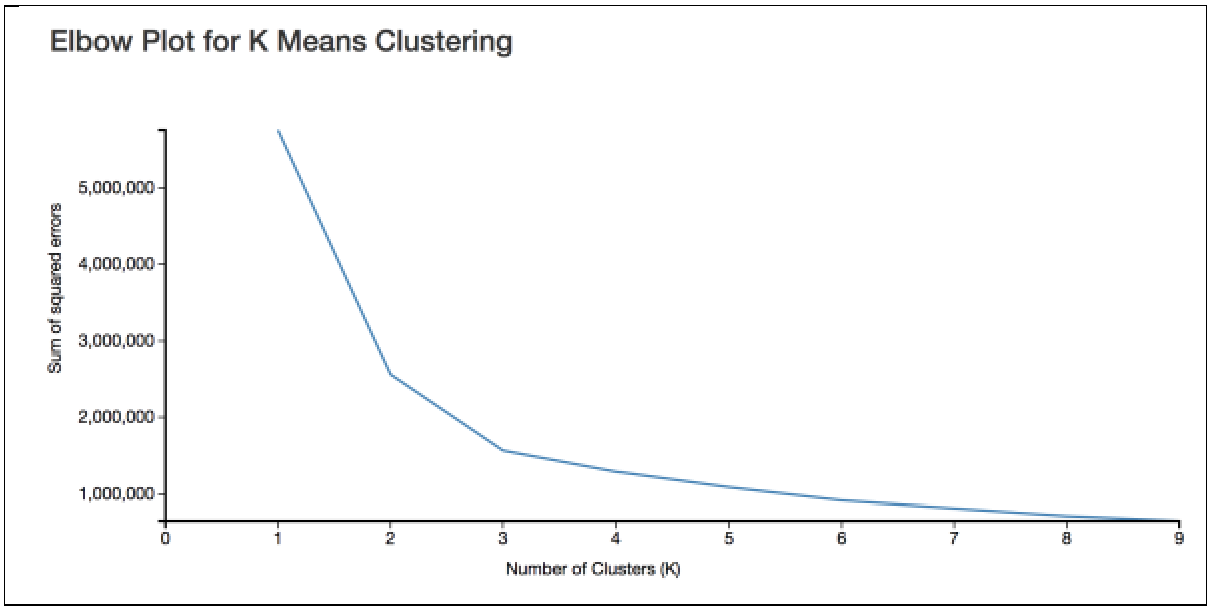
Here in server.py code, we mode random sampling by taking 30% of random samples from input space.

```
# Here we will take 10% samples randomly from input set
inputData = pd.read_csv('my_dengue.csv')
random_samples = inputData.sample(frac=0.3)
```

For stratified sampling, we used K-Means clustering to divide into given into K samples. For finding the optimal K means, we use the elbow method.

```
def elbowKMeans(inputData, n):
    Ks = list(range(1, n))
    km = [KMeans(n_clusters=i) for i in Ks]
    score = [km[i].fit(inputData).score(inputData) for i in range(len(km))]
    df_Ks = pd.DataFrame(Ks)
    df_Ks.columns = ["K"]
    df_score = pd.DataFrame(score).abs()
    df_score.columns = ["k_means_score"]
    sample = df_Ks.join([df_score])
    print(sample.columns)
    sample.to_csv("./static/data/elbow.csv", sep=',')
    return 1
```

The elbow plot is shown in our frontend 'index.html' using d3. Below is the elbow plot.



As we see, we choose 3 as the optimal number of clusters.

```
def stratified_sampling(data_frame, cluster_count, fraction):
    k_means = Kcluster.KMeans(n_clusters=cluster_count)
    k_means.fit(data_frame)

    data_frame['label'] = k_means.labels_

    sample_label = []
    for i in range(cluster_count):
        sample_label.append(data_frame[data_frame['label'] == i].index);
    print("Below are cluster sizes for each of 3 clusters")
    print(len(sample_label[0]))
    print(len(sample_label[1]))
    print(len(sample_label[2]))

    sample_Idx = []
    for i in range(cluster_count):
        sample_Idx.extend(rd.sample(list(sample_label[i]), (int)(fraction*(len(sample_label[i])))))

    data_frame.drop('label', axis=1, inplace=True)
    stratifiedSampleRows = []
    for i in sample_Idx:
        stratifiedSampleRows.append(data_frame.ix[i])

    return pd.DataFrame(stratifiedSampleRows)
```

Now, below is the d3 code snipppet to plot elbow plot

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

        // Add the valueline path.
        svg.append("path")
```

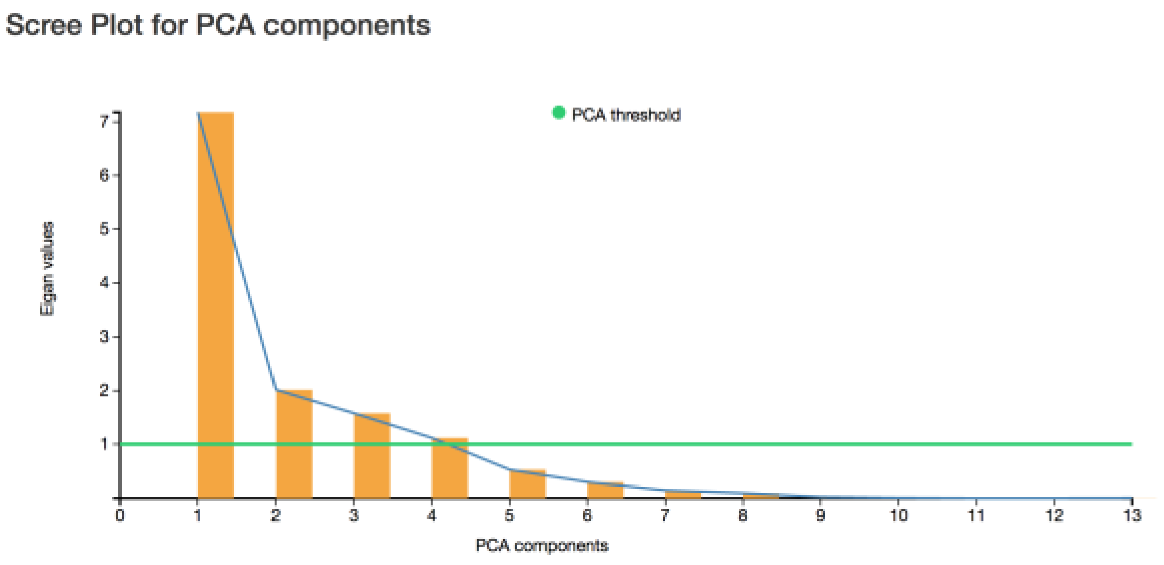
```
            .attr("class", "line")
            .attr("d", valueline(data));

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

          // Add the Y Axis
          svg.append("g")
            .attr("class", "y axis")
            .call(yAxis);

          //put labels
           svg.append("text")
           .attr("transform", "rotate(-90)")
            .attr("y", -100)
            .attr("x", -130)
            .attr("dy", "1em")
            .style("text-anchor", "middle")
            .attr("id", "yLabel")
            .text("Sum of squared errors");


           svg.append("text")
            //.attr("transform", "rotate(-20)")
            .attr("y", 350)
            .attr("x", 350)
            .attr("dy", "1em")
            .style("text-anchor", "middle")
            .attr("id", "xLabel")
            .text("Number of Clusters (K)");
```

Task 2: Dimension reduction

To find the intrinsic dimensionality of data using PCA, we use the scree plot method. Here, first we standardize the input data. Then, find the correlation matrix and find the Eigan vectors and Eigan values. Thus, we will plot Eigan values vs No of PCA components as our scree plot.

```
def createScreeWithEigan(inputSamples, filename):
    X_std = StandardScaler().fit_transform(inputSamples)
    # Eigan values for co-relation matrix
    cor_mat1 = np.corrcoef(X_std.T)
    eig_vals, eig_vecs = np.linalg.eig(cor_mat1)
    y = eig_vals
    x = np.arange(len(y)) + 1
    df_eig = pd.DataFrame(eig_vals)
    df_eig.columns = ["eigan_values"]
    df_pca = pd.DataFrame(x)
    df_pca.columns = ["PCA_components"]
    sample = df_eig.join([df_pca])
    print(sample.columns)
    sample.to_csv("./static/data/" + filename, sep=',')
```

The scree plot is created for both – random samples as well as stratified samples. As we see, we the intrinsic dimensionality of our data will be number of PCA components that have Eigan values greater than 1. So, as we see, we have 4 PCA components greater than 4 i.e. the intrinsic dimensionality of our input data.



The code to create this bar graph is given below –

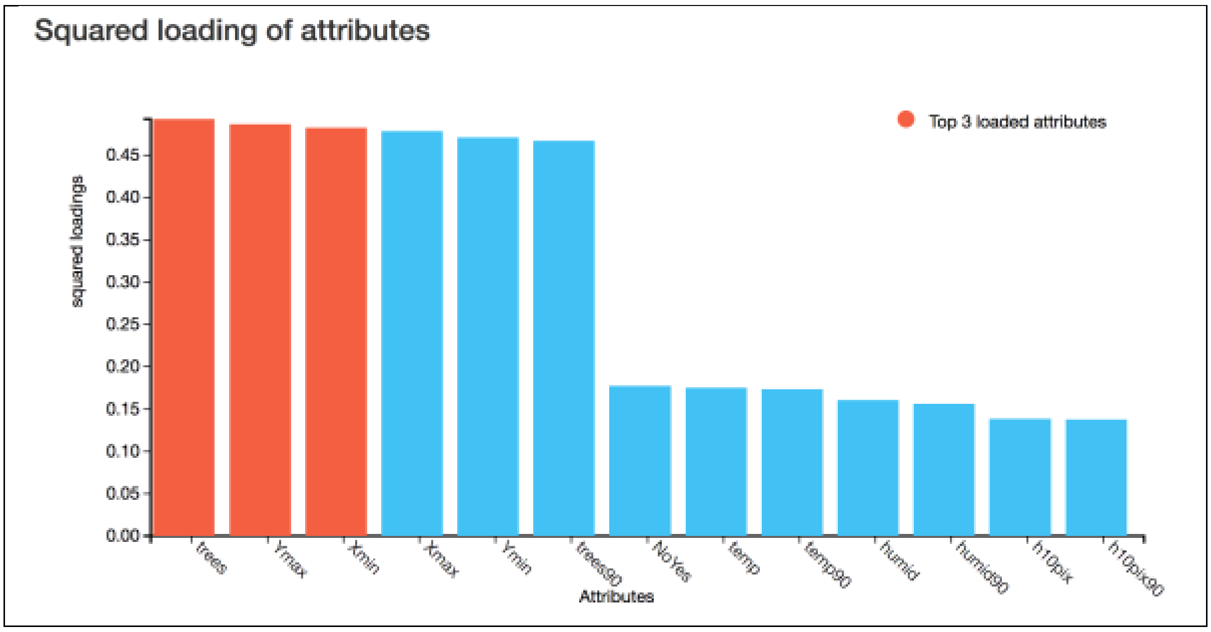
```
            // Add bars
            svg.selectAll(".bar")
              .data(data)
            .enter().append("rect")
              .attr("class", "bar")
              .attr("x", function(d) { return x(d.PCA_components); })
              .attr("y", function(d) { return y(d.eigan_values); })
              .attr("height", function(d) { return height - y(d.eigan_values); })
              .attr("width", 30)
              .attr("fill", "#f4a641");

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

            // Add threshould line
            svg.append("g")
               .attr("transform", "translate(0, "+y(1)+")")
               .append("line")
               .attr("x2", width)
               .style("stroke", "#2ecc71")
```

```
                  .style("stroke-width", "3px");

                // Add legend
                svg.append("circle")
                .attr("cx", "26em")
                .attr("cy", "0em")
        .attr("r","0.4em")
        .style("fill", "#2ecc71");

        // Add label for legend
        svg.append("text")
                .attr("x", "30em")
                .attr("y", "-0.5em")
        .attr("dy", "1em")
        .style("text-anchor", "middle")
        .text("PCA threshold");


}
```

Now, to find the top 3 attributes, we find the sum of squared loadings using PCA and then select the top 3 attributes. Below is the code snippet –

```python
def get_squared_loadings(dataframe, intrinsic, filename):
   std_input = StandardScaler().fit_transform(dataframe)
   pca = PCA(n_components=intrinsic)
   pca.fit_transform(std_input)

   # get the loadings here
   loadings = pca.components_
   squared_loadings = []
   a = np.array(loadings)
   a = a.transpose()

   # find sqaured loadings
   for i in range(len(a)):
      squared_loadings.append(np.sum(np.square(a[i])))

def getTop3attributes(squared_loadings):
   top3 = squared_loadings.head(n = 3)
   return top3['attributes'].values.tolist()
```

The output of this squared loadings in saved in a .csv file which is visualized in the frontend ‘index.html’ using d3 as below –



The d3 code snippet for the squared loadings is given below –

```
// Adds the svg canvas
var svg = d3.select("body")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .attr("id", "scatter")
    .style("margin-left", "20em")
  .append("g")
    .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");
```

```
// Add bars for bar chart
          svg.selectAll(".bar")
           .data(data)
          .enter().append("rect")
           .attr("class", "bar")
           .attr("x", function(d) { return x(d.attributes); })
           .attr("y", function(d) { return y(d.squared_loadings); })
           .attr("height", function(d) { return height - y(d.squared_loadings); })
           .attr("width", 50)
           .attr("fill", function(d, i) { if(i < 3) return "#f45f42"; else return "#41c1f4"});

          // Add legend
          svg.append("circle")
          .attr("cx", "44em")
          .attr("cy", "0em")
    .attr("r","0.5em")
    .style("fill", "#f45f42");

    // Add label for legend
```

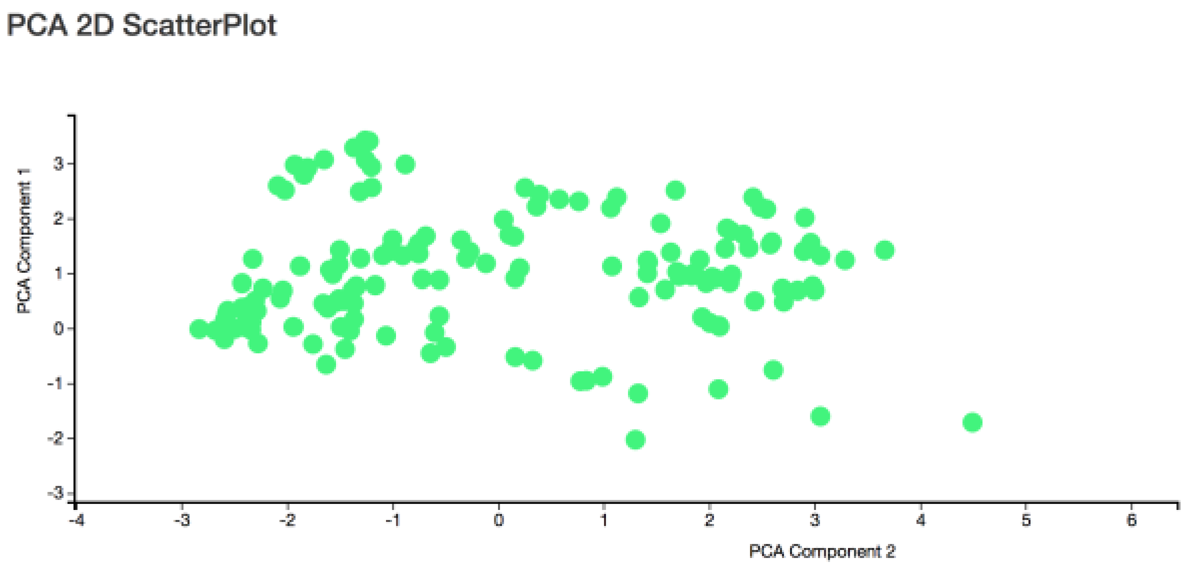
```
        svg.append("text")
                .attr("x", "50.5em")
                .attr("y", "-0.5em")
        .attr("dy", "1em")
        .style("text-anchor", "middle")
        .text("Top 3 loaded attributes");
```

Task 3: Visualizations

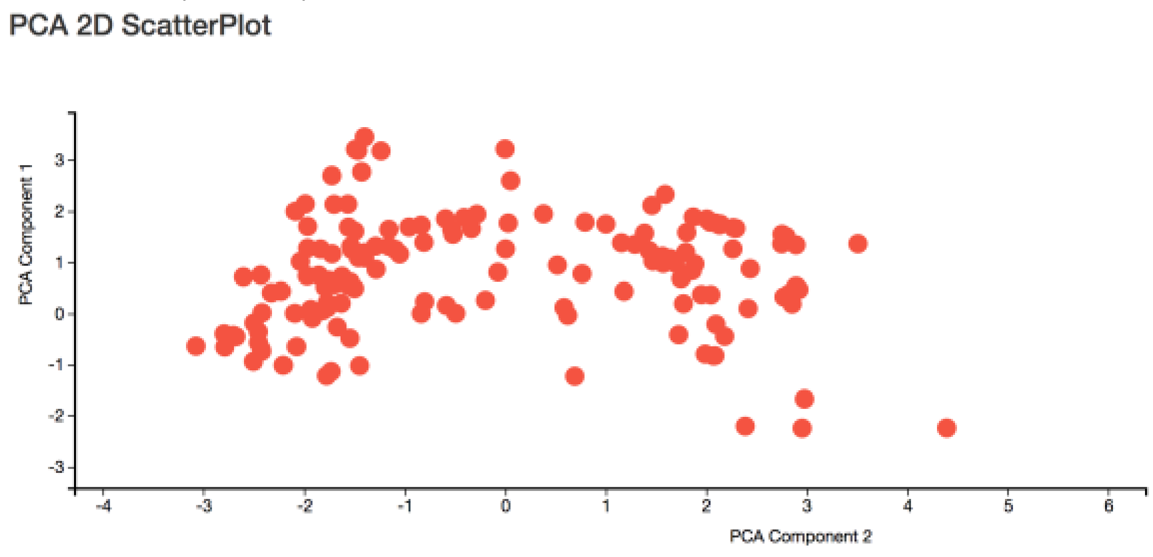
a. Visualize data projected into the top two PCA vectors via 2D scatterplot

   CODE FOR PLOTS IN TAKS 3 a.) and b.) is found in scatterchart.js in source-code/static/js folder.

   [Random Samples PCA plot]

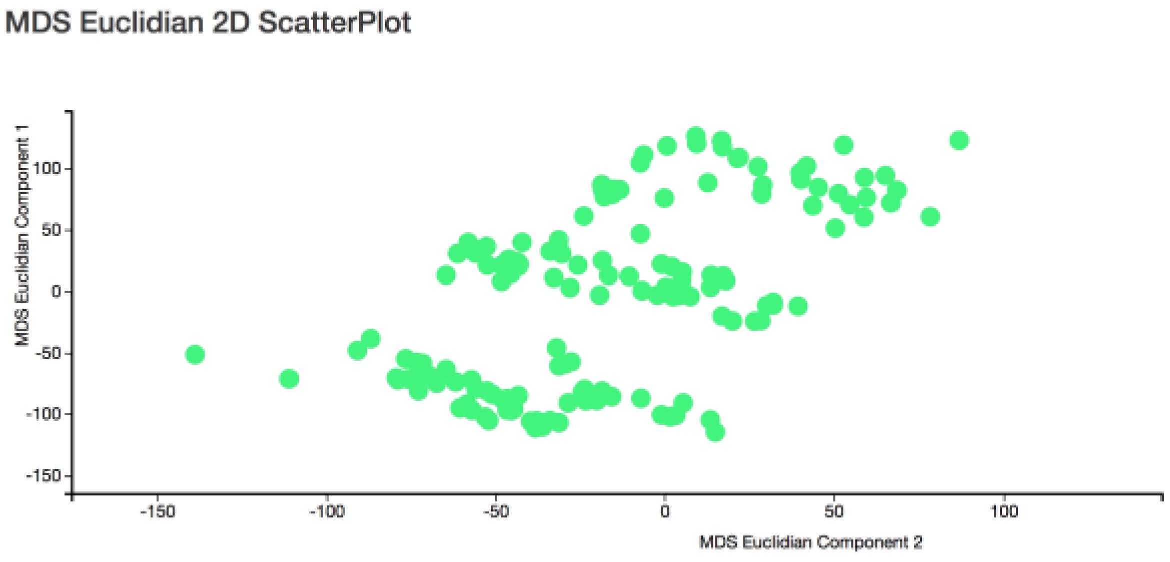


   [Stratified Samples PCA plot]

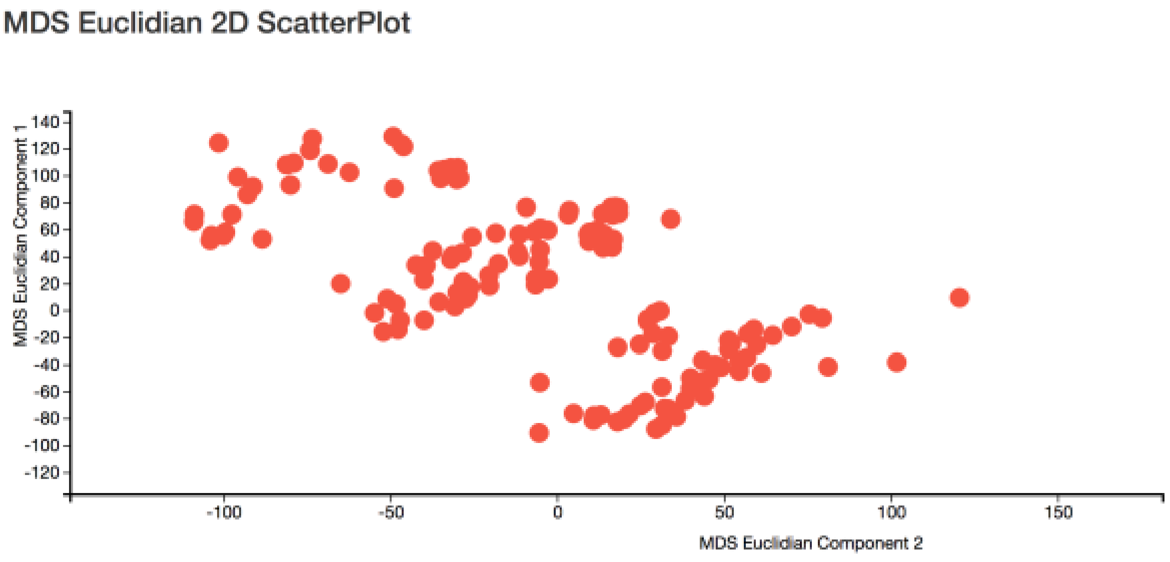


b. Visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots.

[Random Samples MDS plot Euclidian plot]

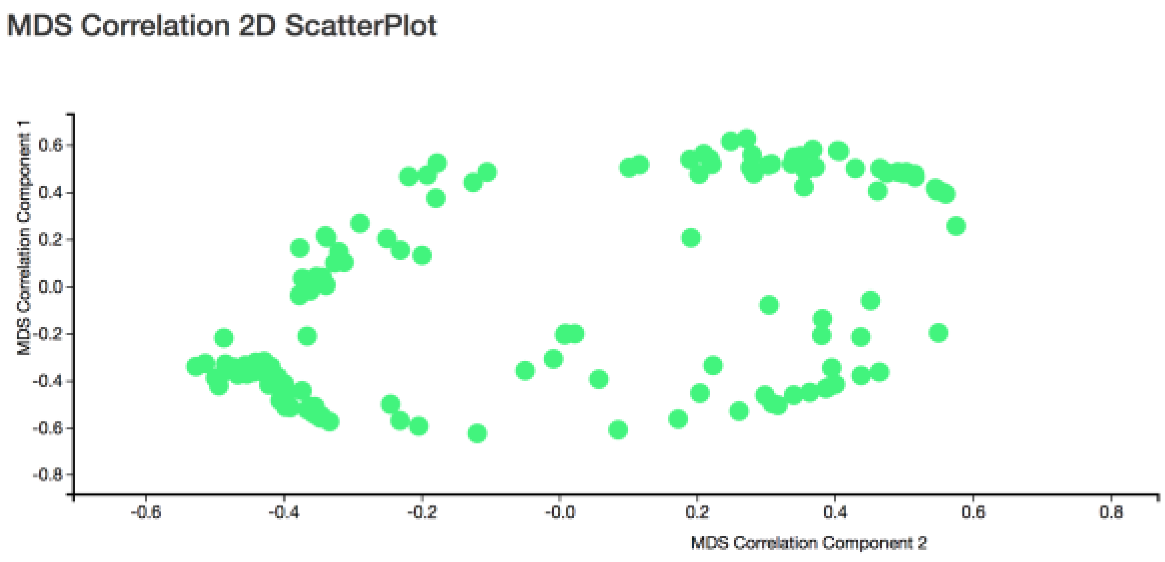


[Stratified Samples MDS plot Euclidian plot]

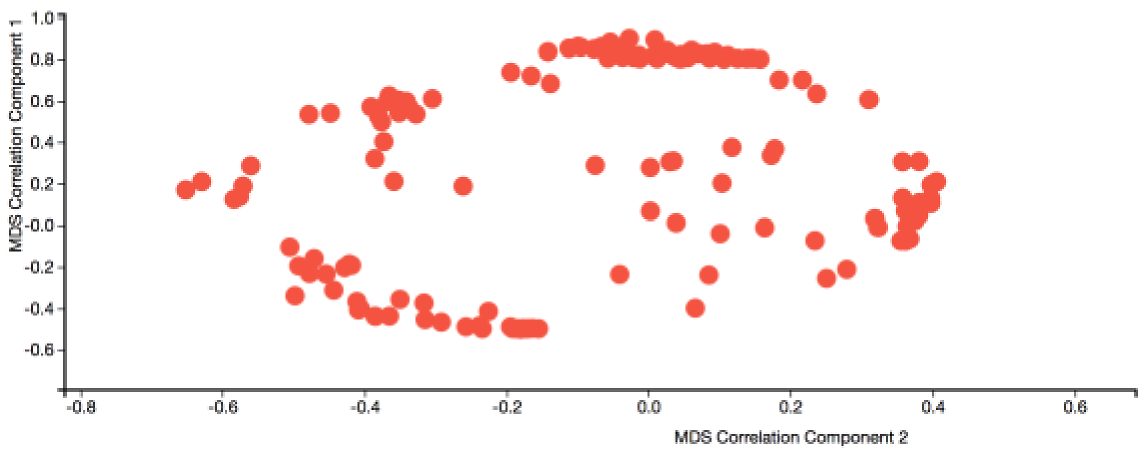


NOTE: We observe that stratified sampling is more sensitive to outliers and draws samples from them too whereas random sampling may ignore them at times if no sample gets drawn out from them.

[Random Samples MDS plot Correlation plot]



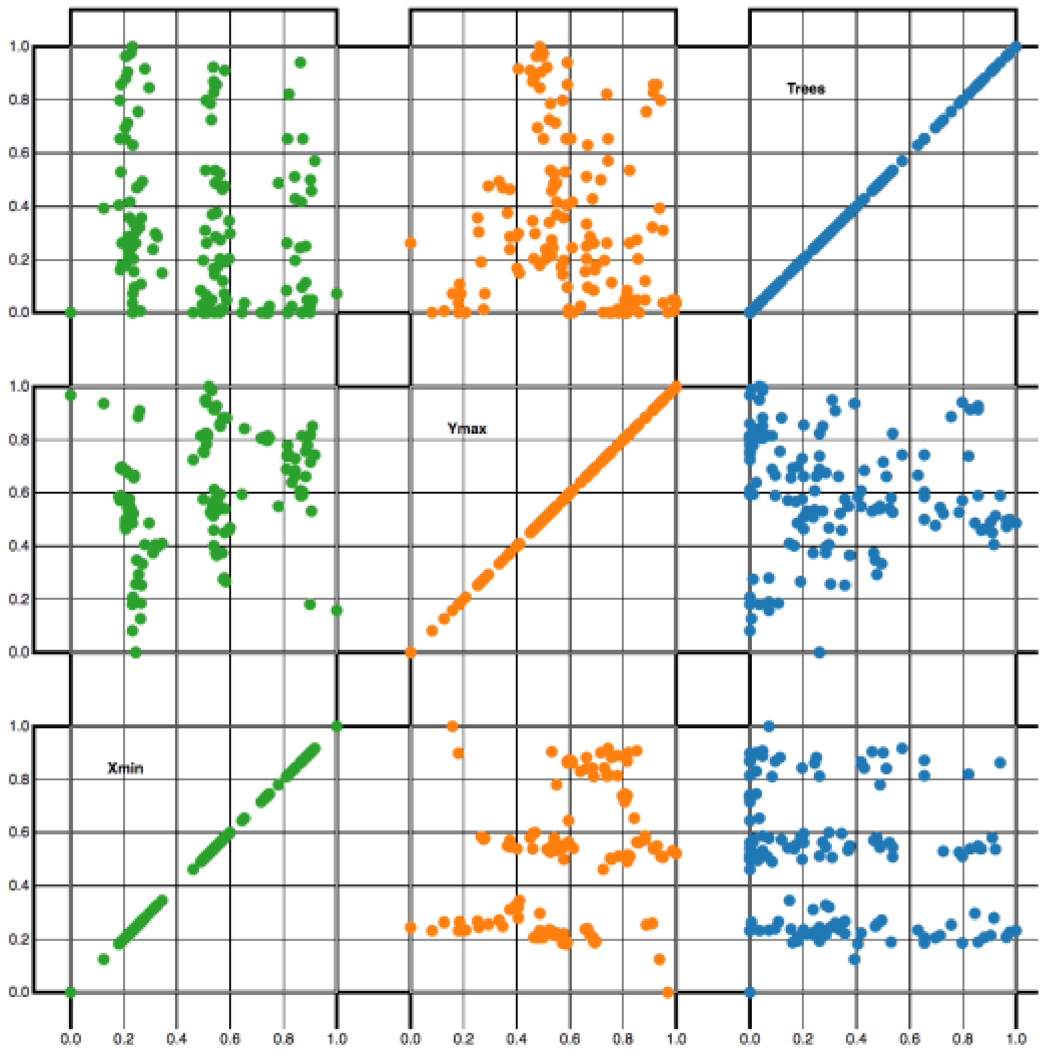
[Stratified Samples MDS plot Correlation plot]



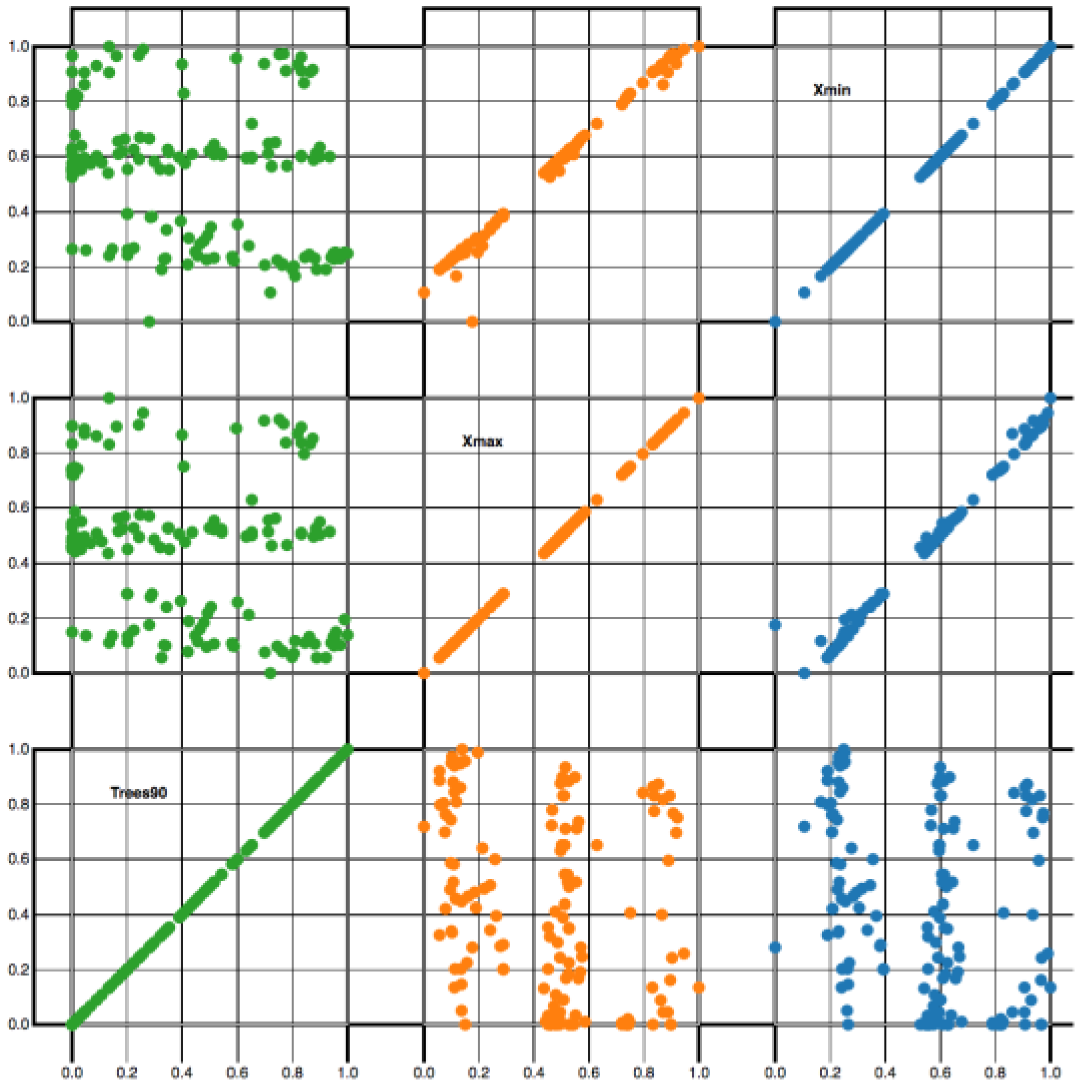
c. Visualize scatterplot matrix of the three highest PCA loaded attributes

CODE FOR PLOTS IN TAKS 3 c is found in scatterplot_matrix.js in source-code/static/js folder.

[Random Samples scatter plot for top 3 samples]



[Stratified Samples scatter plot for top 3 samples]


Xmin
Xmax
Trees90