CSE 564 Visualization

MINI PROJECT 2

Himanshu Agrawal
SBU ID: 112680639

Dataset Description:

NY Students Data (Source: US Census at School)

This facility provides random data samples selected from individuals in the U.S. Census at School population that meet your selected characteristics.
This dataset contains over 60 attributes and the rows vary depending upon how much we can random sample from the website.
I have chosen 12 attributes for this lab as mentioned below:

- Age(years)
- Height(cm)
- Travel time to School
- Score in memory game
- Sleep Hours School night
- Home Occupants
- Doing Homework Hours
- Outdoor Activities Hours
- Video Games Hours
- Computer Use Hours
- Social Websites Hours
- Watching TV Hours

Data Cleanup:

1. Null rows have been removed.
2. Missing values have been imputed using techniques discussed in class.
3. Only numerical columns have been taken for this Lab.

Code Architecture:

1. Server using Flash to run python codes and D3.js has been used in frontend for Visualization.
2. The project contains index.html file to load initial landing page.
3. App.py file contains the python code for data manipulation and running the Flask App.
4. Style.css contains the CSS which describes the styling used in this project.
5. Main.js file contains the code for D3.js visualization implementation and server-side data fetching API calls.

Program Demo YouTube Link: https://youtu.be/mQcHmxteZSg

Task 1 - Data clustering and Decimation

For this task I have to implement 2 types of sampling technique on the data and remove 75% of original data.

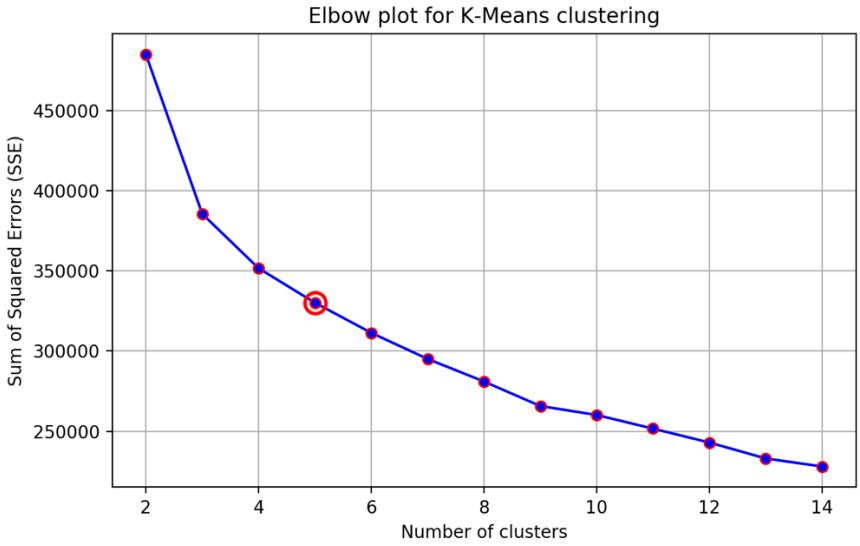
Method 1 – Random Sampling

In this technique I randomly select data(rows) from the feature dataset matrix. This is done using the dataFrame.sample() library function as below:

```
def random_sampling(data):
    random_data = data.sample(frac=0.25, random_state = 1)
    return random_data
```

Method 2 – Stratified Sampling

In stratified sampling we divide the data into separate groups(clusters), called strata. Then, a probability sample (often a simple random sample) is drawn from each cluster. For this we first apply the K-means clustering and find the clusters of data. Using the elbow method to find the k-value, we sample the points based on k-number of clusters.



The k-value for my data from the elbow point as shown above is 5. It means I will now divide the data into 5 clusters and sample my points from those clusters.

Code for using K-means to find clusters:

```
def stratified_sampling(data):
      data_arr = np.array(data)
      sse_array = []
      for i in range(2, 15):
            km = KMeans(n_clusters=i)
            km.fit(data_arr)
            sse_array.append(km.inertia_)
```

Code for Plotting elbow in python:

```
fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(range(2, 15), sse_array, marker='o', markeredgecolor='r', color='b')
    # Mark the Elbow Point
    ax.plot(5, sse_array[3], marker='o', markersize=12, markeredgewidth=2,
markeredgecolor='r',markerfacecolor='None')
    plt.grid(True)
    plt.xlabel('Number of clusters')
    plt.ylabel('Sum of Squared Errors (SSE)')
    plt.title('Elbow plot for K-Means clustering')
    plt.xticks(range(2, 15, 2))
    plt.draw()
```

Code for Sampling points from above:

```
    km = KMeans(n_clusters=5)
    km.fit(data_arr)
    data['Label'] = km.labels_
    cluster_sizes = np.bincount(km.labels_)
    stratified_sampling_results = pd.DataFrame(columns=data.columns)
    for i in range(5):
            cluster_size = cluster_sizes[i]
            cluster_records = data[data['Label'] == i]
            sample_size = int(cluster_size * 0.25)
            stratified_sampling_results = pd.concat([stratified_sampling_results,
cluster_records.iloc[random.sample(range(cluster_size), sample_size)]])
    return stratified_sampling_results
```

Task 2 – Dimension Reduction

In this task we have to find the intrinsic dimensionality of the data using PCA(Principal Component Analysis). To do this we first find the PCA of the data and plot the Eigenvalues (Variance) and the PCA components.

The code is:

```
def calculate_PCA(data):
      pca = PCA()
      pca.fit(data)
      pca_results = pca.explained_variance_ratio_
      loadings = np.sum(np.square(pca.components_), axis=0)
      indices_of_top_3_attributes = loadings.argsort()[-3:][::-1]
      top_two_components = pca.components_[:2]
      return pca_results, indices_of_top_3_attributes, top_two_components
```

Now we make a screeplot by sending the pca_results to front-end and plot it using D3.js.

ScreePlot D3.js Code:

First we define our SVG element, and set the margins, scaling and axis:

```
    d3.select('#chart').remove();
     var margin = {top: 100, right: 100, bottom: 100, left: 100},
     width = 1000 - margin.left - margin.right,
     height = 700 - margin.top - margin.bottom;

     var x = d3.scale.ordinal().domain(data_cum.map(function(d,i){return
i+1;})).rangeRoundBands([0, width],.1);
     var y = d3.scale.linear().domain([0, d3.max(data_cum)]).range([height, 0]);
     var xAxis = d3.svg.axis().scale(x).orient("bottom");
     var yAxis = d3.svg.axis().scale(y).orient("left");
     var color = d3.scale.category10();
     // Add an SVG element with the desired dimensions and margin.
     var svg = d3.select("body").append("svg")
           .attr("id", "chart")
           .attr("class","svgContainer")
           .attr("width", width + margin.left + margin.right)
           .attr("height", height + margin.top + margin.bottom)
           .append("g")
           .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

Then we plot the bars, which represent the individual PCA component eigenvalues.

```
    svg.selectAll('rect')
          .data(data_eigen)
          .enter()
          .append('rect')
          .attr('x', (d,i) => x(i+1))
          .attr('y', d => y(d))
          .attr('width', (width / data_cum.length) - 5)
          .attr('height', d => height - y(d))
          .attr('fill', 'green');
```

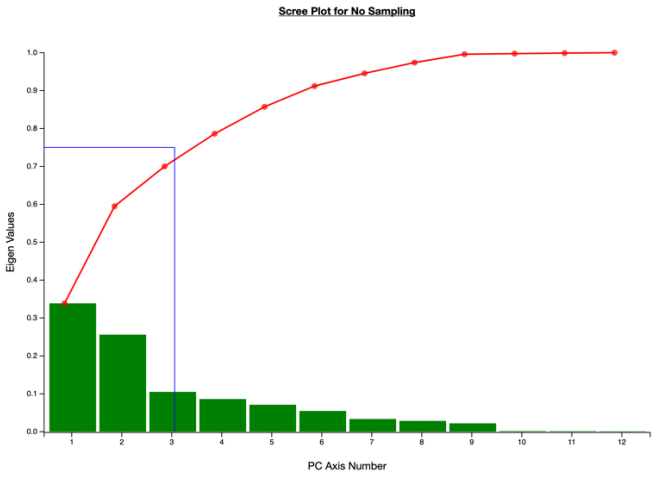
Then I plot a line chart representing the cumulative explained variance.

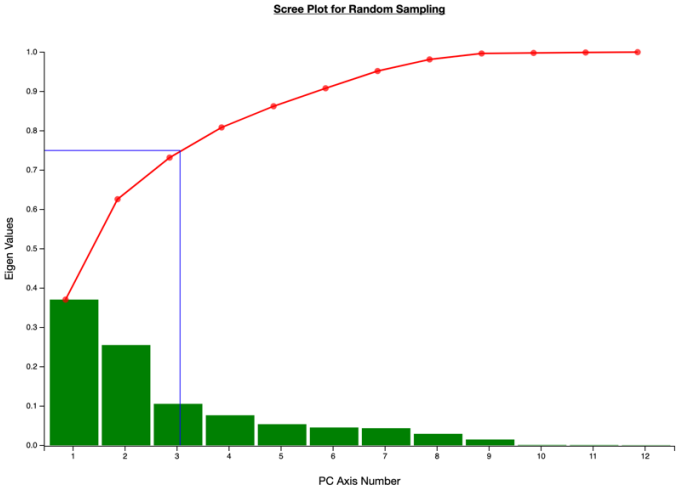
```
    var line = d3.svg.line()
        .x(function(d,i) { if (i == 2) { markerX = x(i+1)+20; markerY = y(d)}return
x(i+1)+20;})
        .y(function(d) { return y(d); })

    //Create Line Path
    svg.append("path")
        .attr("class", "line")
        .attr("d", line(data_cum))
        .attr("transform", "translate(0,0)")
        .attr("fill","none")
        .attr("stroke","red")
        .attr("stroke-width","2px");
```

Then I plot circles/dots on the line which when Hovered Over will show the cumulative values.

```
    svg.selectAll("circle")
        .data(data_cum)
        .enter()
        .append("circle")
        .attr("cx", (d,i) => x(i+1)+20)
        .attr("cy", d => y(d))
        .attr("r", 4)
        .attr("transform", "translate(0,0)")
        .attr("fill","red")
        .on("mouseover",handleMouseOver)
        .on("mouseout", handleMouseOut)
```

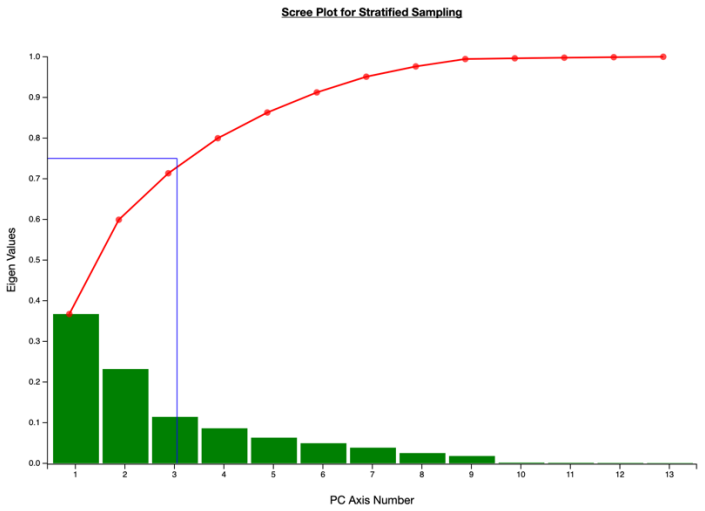
The axis labelling and the significant lines D3 code is simple and can be seen in the project files.
The output of this visualization is a Scree Plot as shown below for all Three Types of data:






```
Top 3 Attributes:

['Score_in_memory_game', 'Home_Occupants', 'Watching_TV_Hours']

127.0.0.1 - - [07/Apr/2020 18:37:41] "GET /noSamplingScree HTTP/1.1" 200 -
```

```
Top 3 Attributes:

['Travel_time_to_School', 'Outdoor_Activities_Hours', 'Doing_Homework_Hours']

127.0.0.1 - - [07/Apr/2020 18:38:17] "GET /randomSamplingScree HTTP/1.1" 200 -
```




```
Top 3 Attributes:

['Ageyears', 'Watching_TV_Hours', 'Video_Games_Hours']

127.0.0.1 - - [07/Apr/2020 18:38:48] "GET /stratifiedSamplingScree HTTP/1.1" 200 -
```

Using the Elbow method we can find the intrinsic dimensionality of the data. Also, by using the cumulative explained variance of 75% we can find the dimensionality. Here, using both methods I find my data dimensionality to be 3 as can be seen in all the 3 plots. This is also portrayed by the cumulative explained variance represented by the significant blue line in the Plot.

Thus, the dataset can best be represented using five principal components.

To obtain the 3 highest PCA loaded attributes :

- We need to calculate the squared loadings. Each element of eigen vector represents the contribution of a given variable to a component. Loadings are these eigen vector elements or the correlation between variables and principal components.
- Then sort these squared loadings and take the top n components. In our case, it is 3 from PCA scree plot to do further data analysis.
- The 3 attributes are printed in the Python terminal which can be seen as above.

Task 3 – Visualization of Data

I. Visualizing Top 2 PCA vectors using Scatterplot

Top two PCA vectors are those that correspond to highest eigen values. Here scikit library is used to calculate PCA components with number of components = 2. Visualization of top 2 PCA vectors are done for original data, randomly sampled data and stratified sampled data. Code for PCA is shown below.

```
def pca_Scatter(data):
      pca = PCA(n_components=2)
      X = data
      principalComponents = pca.fit_transform(X)
      return principalComponents
```

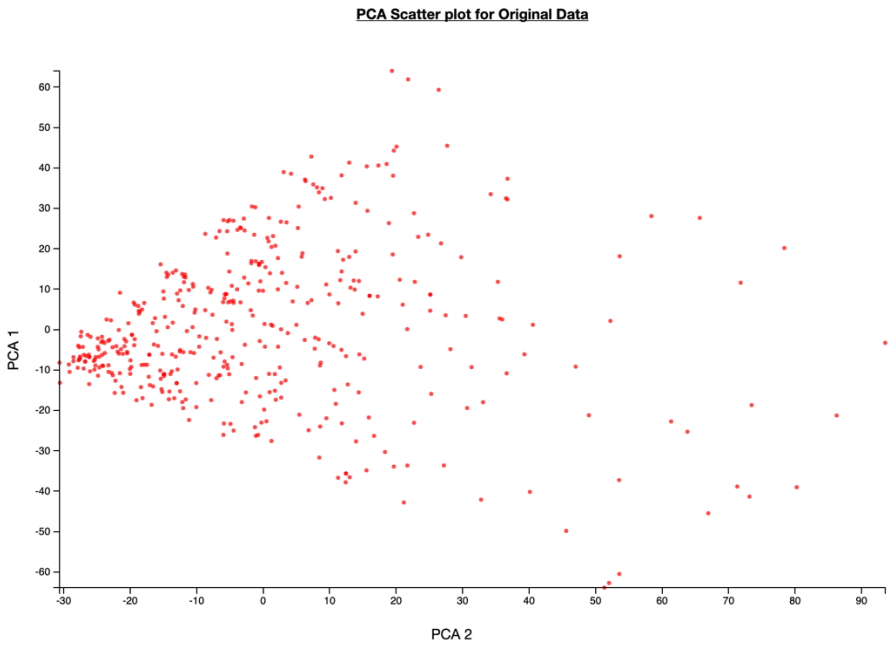
The parameter data will be passed depending on which sample PCA values we want to find.

The D3.js code for Visualizing the Scatter plot which will be used in Task 3 I and Task 3 II is like below:

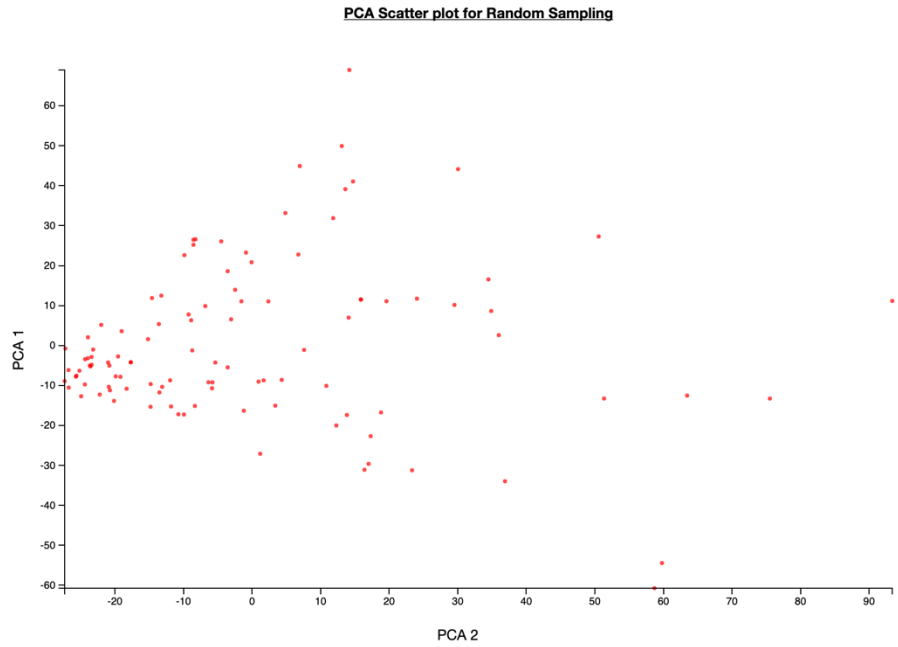
```
    d3.select('#chart').remove();
    var margin = {top: 100, right: 100, bottom: 100, left: 100},
        width = 1000 - margin.left - margin.right,
        height = 700 - margin.top - margin.bottom;
```

```
 5      var x = d3.scale.linear().domain([d3.min(data,d => d[0]),d3.max(data,
d=>d[0])]).range([0,width]);
 6      var y = d3.scale.linear().domain([d3.min(data,d => d[1]),d3.max(data,
d=>d[1])]).range([height, 0]);
 7      var xAxis = d3.svg.axis().scale(x).orient("bottom");
 8      var yAxis = d3.svg.axis().scale(y).orient("left");
 9      var color = d3.scale.category10();
10
11      var svg = d3.select("body").append("svg")
12            .attr("id", "chart")
13            .attr("class","svgContainer")
14            .attr("width", width + margin.left + margin.right)
15            .attr("height", height + margin.top + margin.bottom)
16            .append("g")
17            .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
18      // Plot scatter dots
19      svg.selectAll("circle")
20          .data(data)
21          .enter()
22          .append("circle")
23          .attr("cx", (d,i) => x(d[0]))
24          .attr("cy", (d,i) => y(d[1]))
25          .attr("r", 3)
26          .attr("fill","red")
```

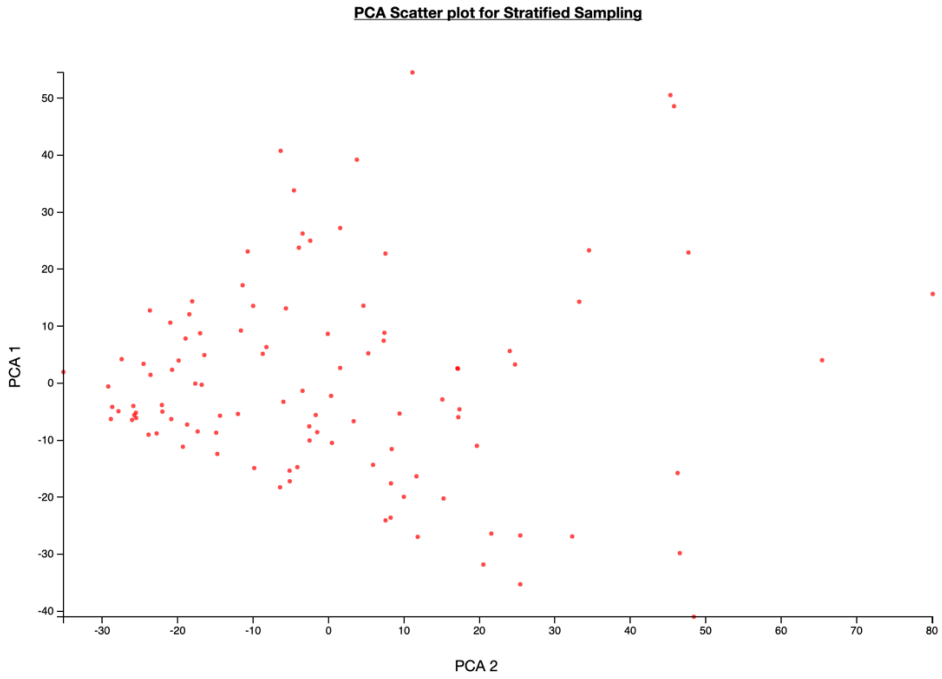
Other part of code can be seen in the project file.
Below, I show the PCA scatter plot for Top 2 attributes of all 3 types of data samples:



The above PCA plot for Original data we can see that the data is spread from a single point and closer to PCA1 axis. We see a condensed group of points towards the left explaining the high weight of PCA component 1.



The above plot for random sampling is similar to original, just the number of samples has been reduced. And we see a small group of points towards the left explaining the high weight of PCA component 1.



These plots show that the PCA technique can be used to reduce the dimensions of the data without losing much information. The data is transformed such that component with the maximum variation is plotted on the Y-axis and component with the second-best variation is plotted on the X-axis.

II. Visualizing the data using MDS(Euclidean and Correlation distance) using Scatterplot

The code for plotting the scatter plot remains the same as the previous part. Just the interpretation of plot will change, and we will use MDS(Multidimensional Scaling).
The MDS technique is employed to show its dimension reduction capabilities. Both Euclidean distance and correlation distance methods are used on both random and adaptive samples from task 1. MDS and pairwise_distance form sklearn.metrices are used to calculate the pair-wise distance and the MDS. MDS finds a set of vectors in p-dimensional space such that the matrix of Euclidean distances among them corresponds as

closely as possible to some function of the input matrix according to a criterion function called stress. The plot depicts the same behavior as all the points are closely packed together. The Euclidean distance groups the points according to the Euclidean distances while the correlation method tries to put variables with high positive correlations near each other, and variables with strong negative correlations far apart.

For MDS using Euclidean distances: It is calculated using the python code as below:

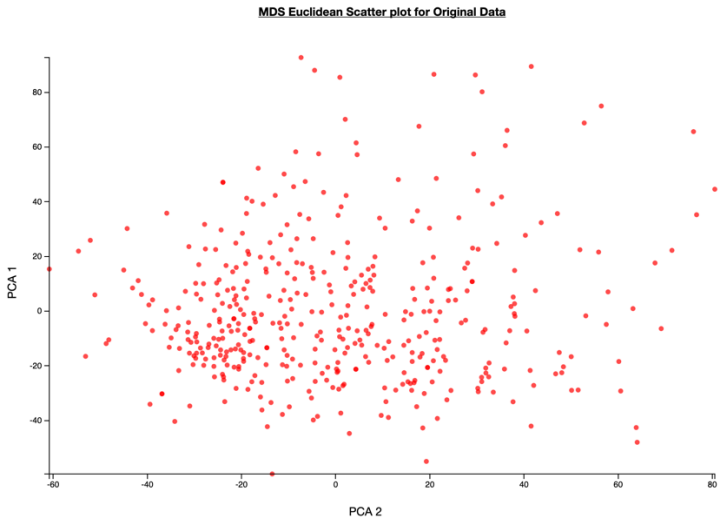
```
def calculate_MDS_Euclidean(data):
      mds_data = MDS(n_components=2, dissimilarity='precomputed')
      similarity = pairwise_distances(data, metric='euclidean')
      X = mds_data.fit_transform(similarity)
      return X
```

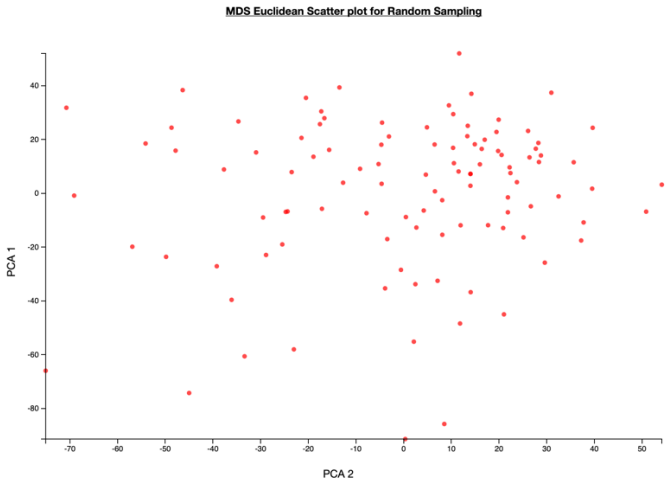
For MDS using Correlation distances: It is calculated using the python code as below:

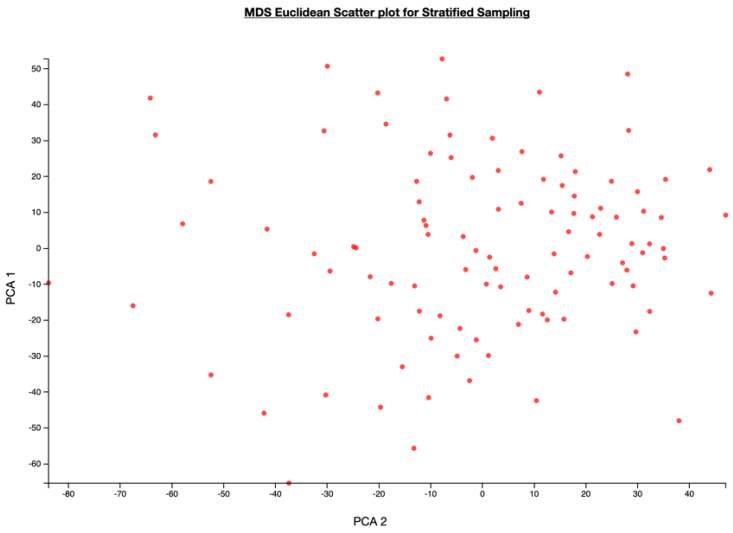
```
def calculate_MDS_Correlation(data):
      mds_data = MDS(n_components=2, dissimilarity='precomputed')
      similarity = pairwise_distances(data, metric='correlation')
      X = mds_data.fit_transform(similarity)
      return X
```

The input data parameter is sent depending on which sample we want to be visualized.

The plots of these distances are as below:

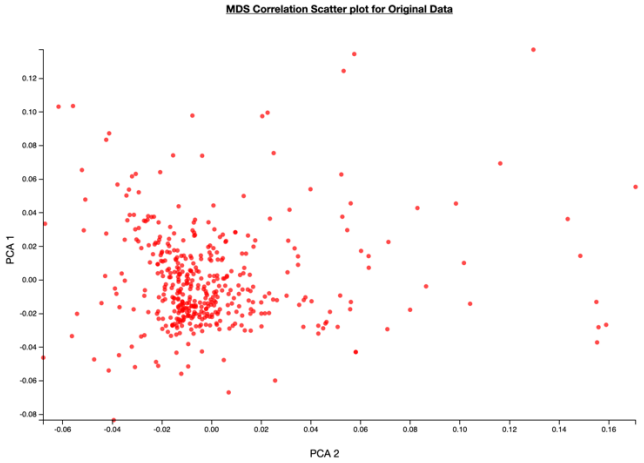


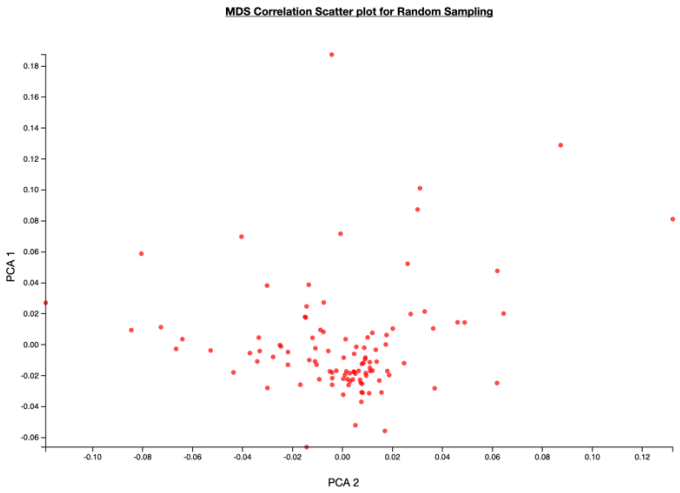


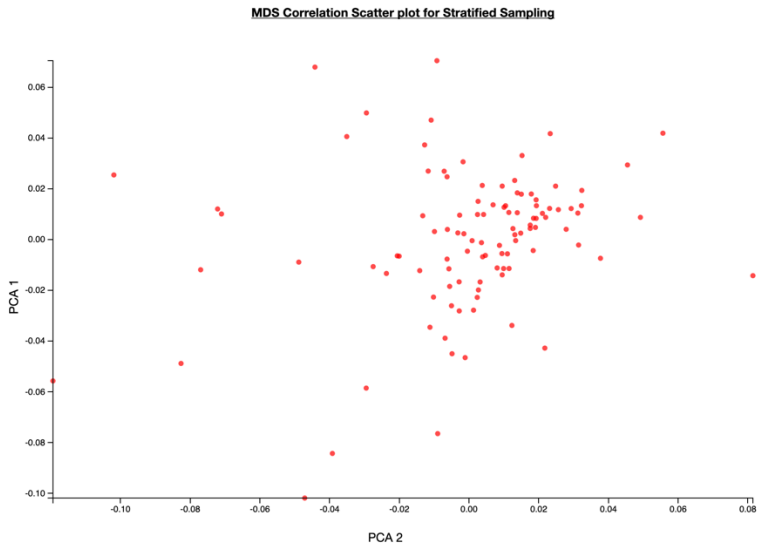


The MDS Euclidean distance plot are as shown. The Points are spread all over, and points with high correlation are placed nearby. gives points which seem equidistant from both the axis. This shows that both components contribute to MDS.

MDS with Correlation Distance plots are as below:





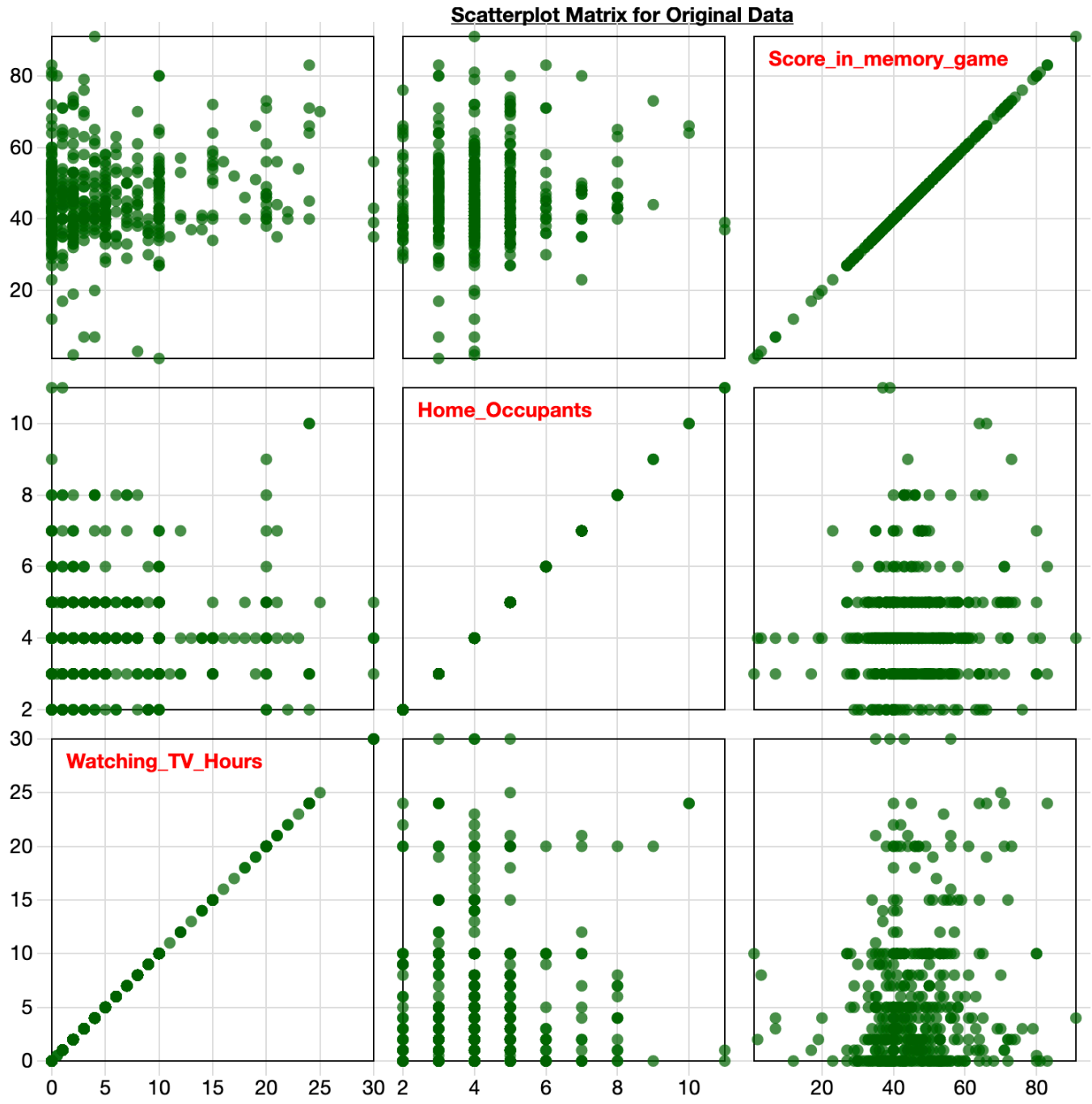


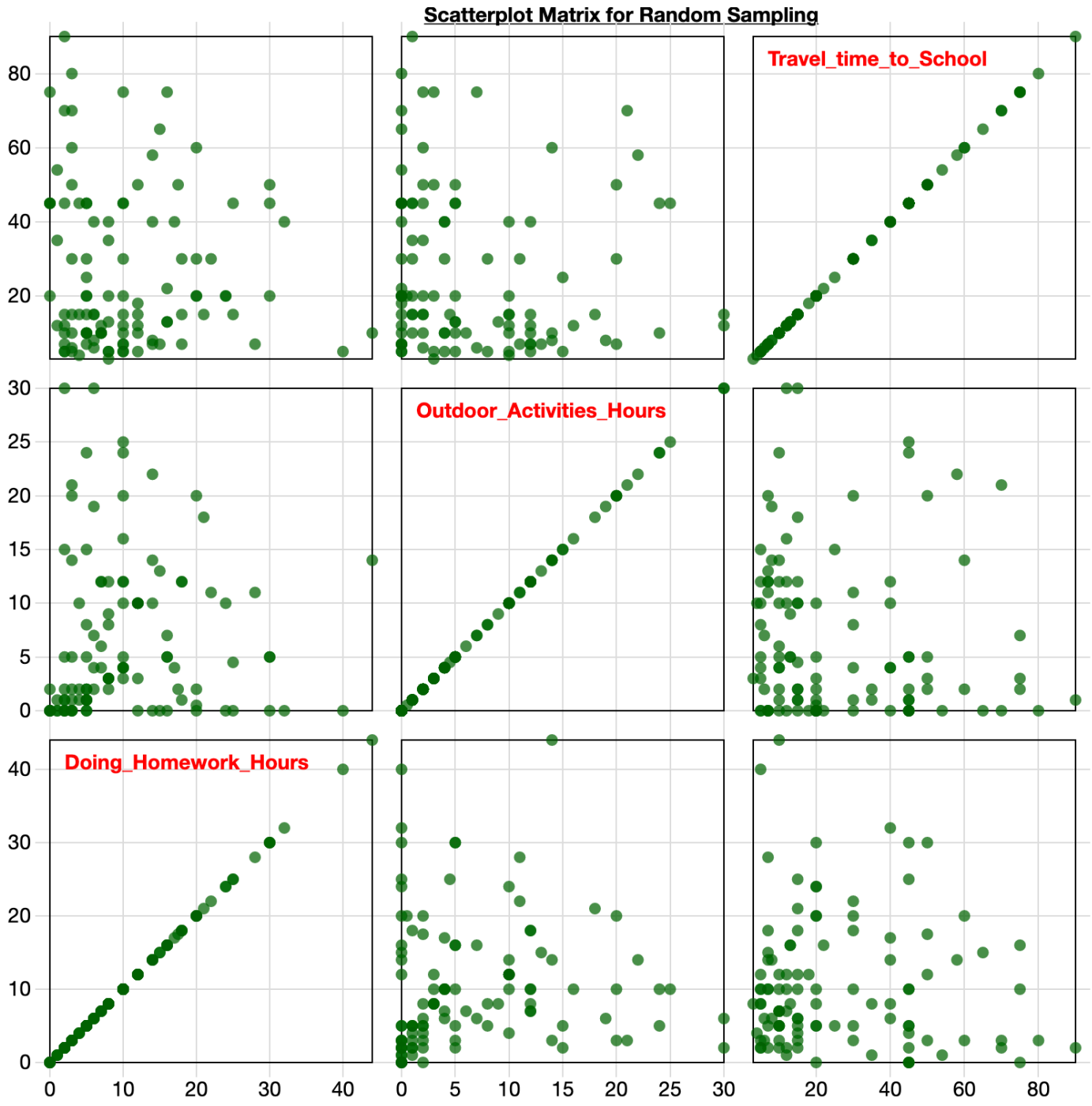
In the MDS Correlation plots, we can see that in all plots the points form some concentrated area, showing that many points in the data are closely correlated. While points which are spread out have less correlation. We can say that both components contribute in MDS Correlation Distances plot.

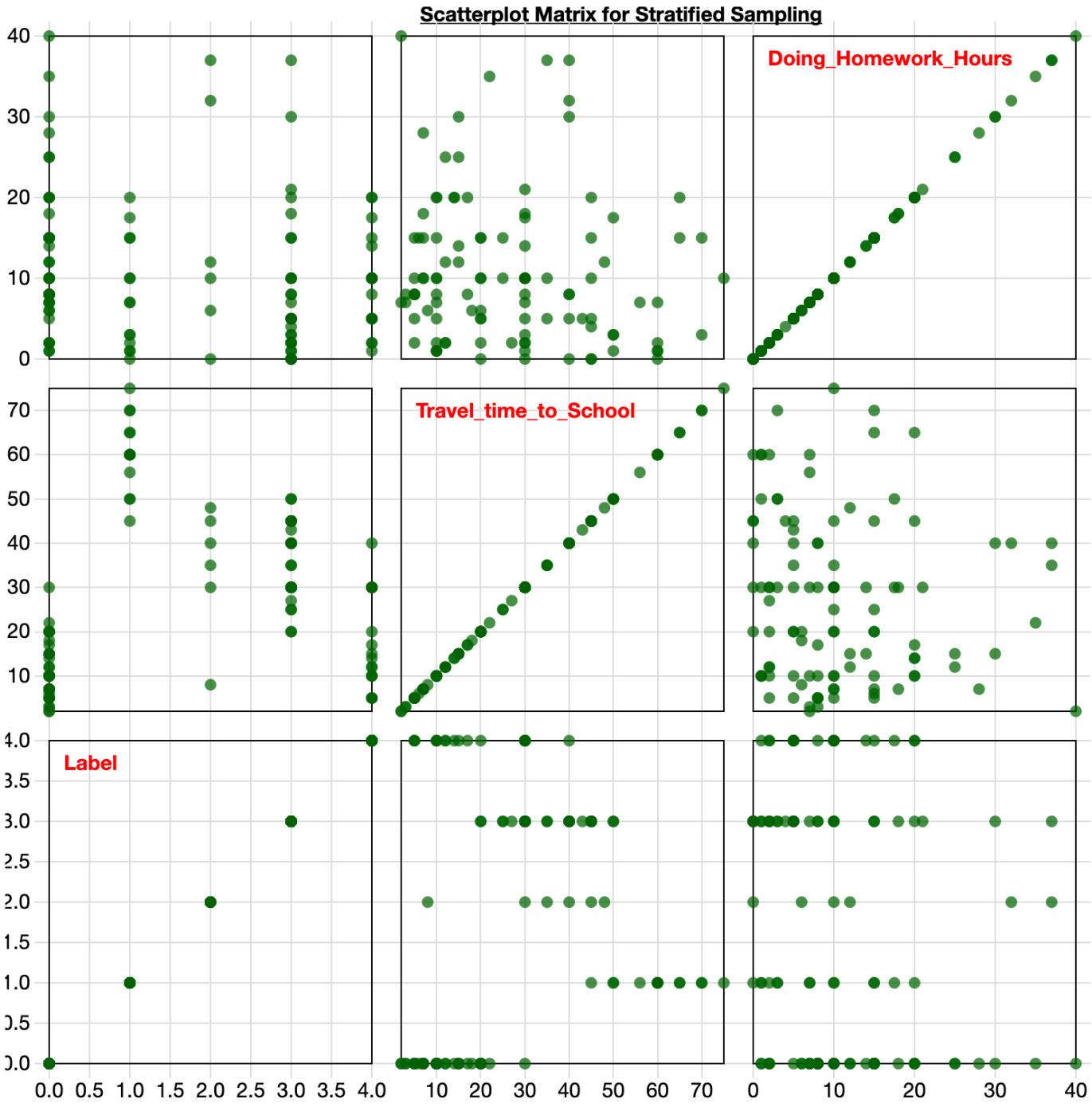
II. Visualizing the Top 3 loaded attributes using Scatterplot Matrix

A scatter plot matrix is a grid (or matrix) of scatter plots used to visualize bivariate relationships between combinations of variables. Each scatter plot in the matrix visualizes the relationship between a pair of variables, allowing many relationships to be explored in one chart. There are 9 sub plots showing the variance and covariance between the parameters placed on x-axis and y-axis.

The scatterplot matrix of all three types of data with their respective highest 3 loaded attributes are shown below.
The plots are pretty symmetric to diagonal. And some points in some cells coincide, probably because they are similar/correlated.




Scatterplot Matrix for Random Sampling
Travel_time_to_School
Outdoor_Activities_Hours
Doing_Homework_Hours
80
60
40
20
30
25
20
15
10
5
0
40
30
20
10
0
0 10 20 30 40
0 5 10 15 20 25 30
20 40 60 80



******** END OF REPORT ********