Visualization Lab 2

Name: Viraj Kamat
SBU-ID: 112818603

Task 1

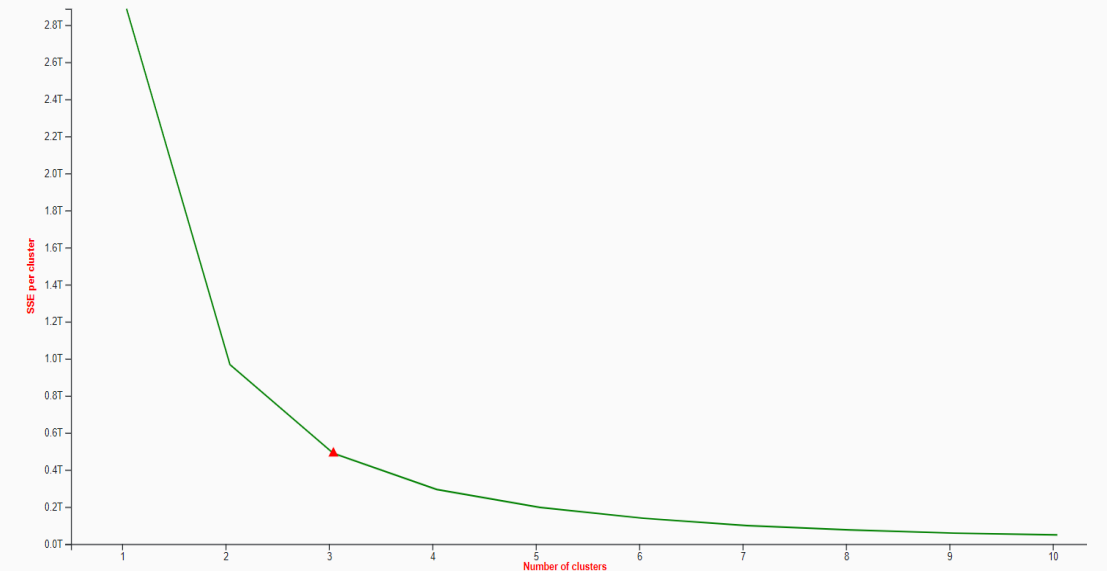
Creating Randomized Data :

The randomized data was created using the pandas module. Essentially the data used was the same as that in the first lab and consisted of cars and their attributes such as model, popularity, price, etc. The data was loaded using a pandas dataframe module and then the **.frac** function was applied with a fraction of .3, this yielded approximately 30% of the data randomly selected from the dataset.

This dataset was saved in a csv file to be used later.

Creating Stratified Data :

In order to create stratified data I used sklearn **kmeans** feature. In order to first cluster the data as per the kmeans algorithm I had to find out the optimum number of clusters, this was done with the '**elbows method**'. The diagram below demonstrates the elbow curve.



It is essentially a plot of the sum of squared errors per cluster plotted against each cluster.As can be seen from the above graph, the elbow appears at 3 clusters, thus the kmeans clustering algorithm was run on the entire cars dataset and thereby created 3 clusters.

We needed to select data from these clusters to create our stratified sample, with the total datapoints adding upto 30% of the data. Hence datapoints were proportionately picked from each cluster with the formula :

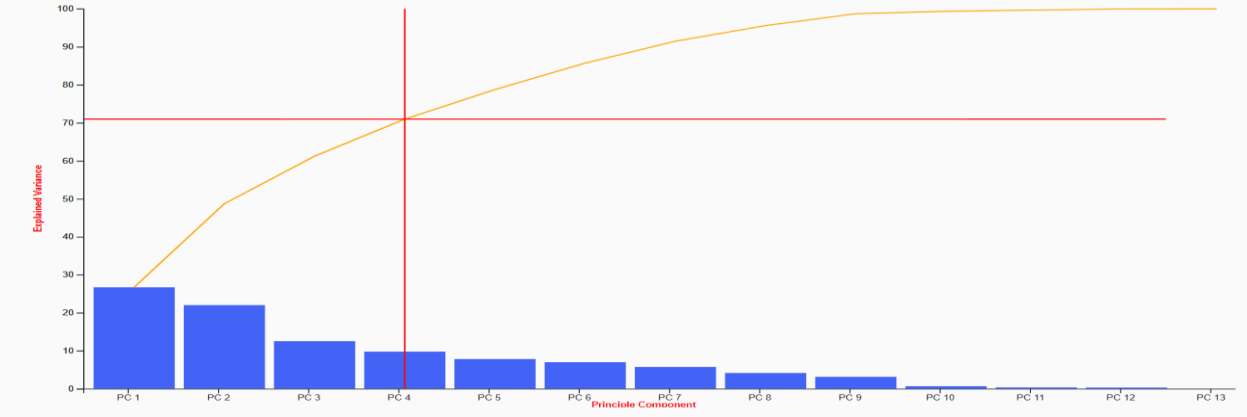
Fraction_per_cluster = (records_in_cluster/total_records).3/records_per_cluster*

Task 2

In order to perform the Principal component analysis I used the sklearn.decomposition PCA feature. I fed the 3 datasets (stratified,randomized, complete) into the PCA function and retrieved the explained variance per component – which is the intention of PCA. The code below demonstrates how this was done along with the scree plot, note that the data was first scaled in order to perform the PCA

```
scaler =  MinMaxScaler(feature_range=[0, 1]) #StandardScaler()
x_train = scaler.fit_transform(features)
x_test = []
if n_components != False :
    myreducer = PCA(n_components=n_components)
else :
    myreducer = PCA()
x_train = myreducer.fit_transform(x_train)
return myreducer,x_train,columns
```

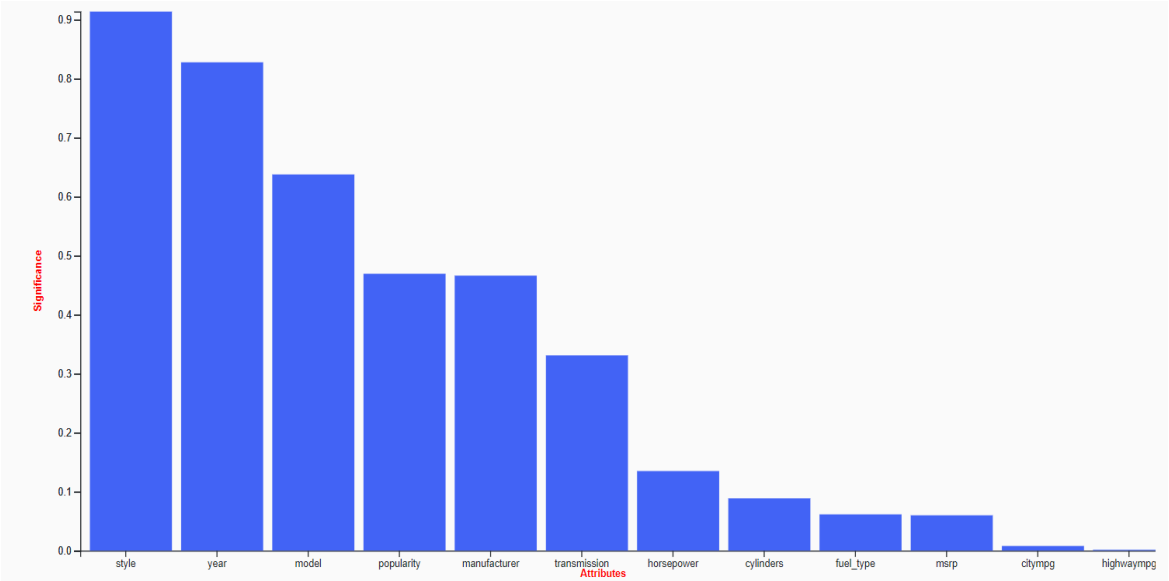
What was noted in the randomized and complete dataset that the 4 principal components can accurately reveal the explained variance of the dataset, in the case of stratified data however around five components are needed. Below is a screen shot of the explained variance per component, the yellow line displaying the cumulative explained variance per component. The red line marking the intrinsic dimensionality.



The next part of task 2 asked us to find the top 3 attributes, the following steps were used for this process :

1. In step 1 I found the number of components that could explain the variance of the dataset to be 4, based on that I found the eigen vector of each of the components per attribute.

2. Each element of the eigen vector denotes the contribution given by an attribute to a component. Thus, the we fetched the eigen values per attribute for all the components and performed a sum of squares of those values. The output we get is the significance of the attribute per component.
3. The most significant variables were fetched by sorting the value we obtained in the above steps and fetching only the top 3 attributes, the diagram below denotes the top 3 attributes in complete dataset based on the above steps.



The **style, year and the model** were found to be the most significant attributes of the car. This was also true for the randomized dataset, however for the stratified dataset the most important attributes were found to be **style, year & popularity**.

The code snippet below shows how this computation was performed:

```
loadings = myreducer.components_.T  # * np.sqrt(myreducer.explained_variance_)
loadings = pd.DataFrame(loadings, columns=['PC1', 'PC2', 'PC3', 'PC4'])
loadings = np.sum(np.square(loadings), axis=1)
```

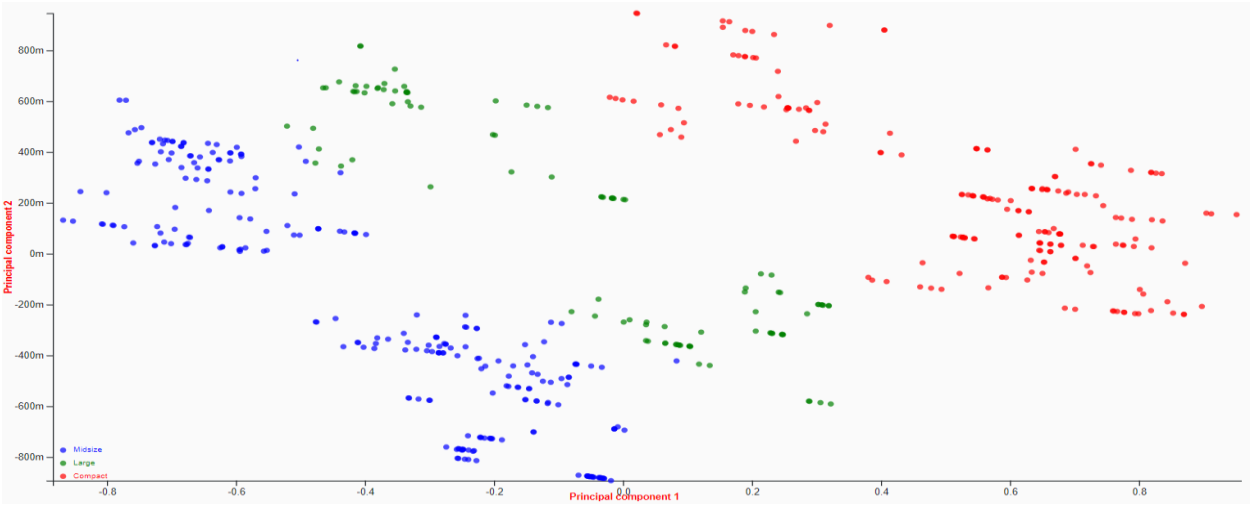
Task 3

In step 1 of task 3, to plot the scatter plot of the top 2 PCA vectors I first performed a PCA with the number of components as 2. The scatterplot could potentially show us how the datapoints cluster.

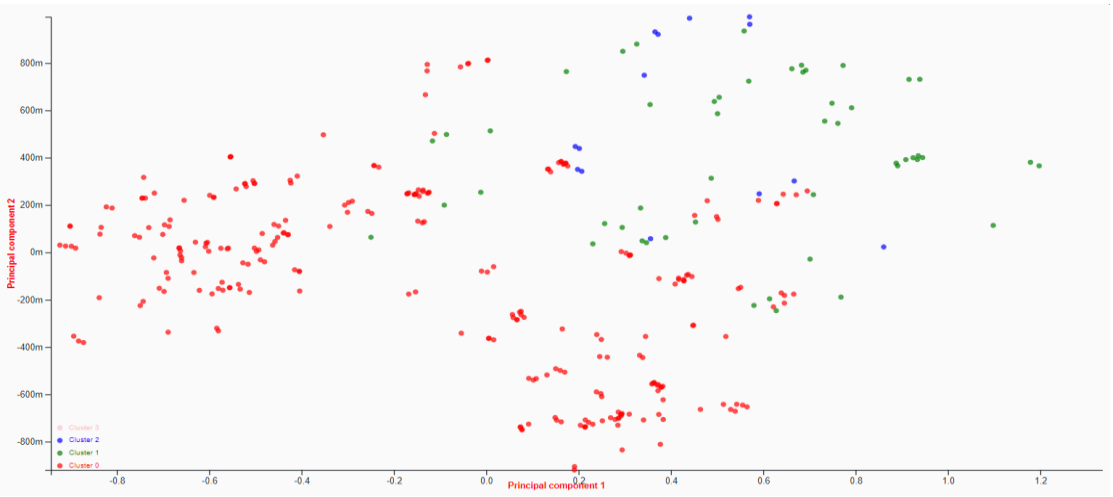
Thus, in the case of randomized data and complete data I used the style of the case to potentially identify those cars that were similar and identify whether they cluster together, in the case of stratified data however I preserved the cluster number of each datapoint, I then went on to plot the top two pca vectors, the code below shows how this computation was performed in the python backend server

```
if index in ["completedata","randomizeddata"] :
    labels = pd.DataFrame(dataframe['size'], columns=['size'])
    labels = pd.DataFrame(data_loader.size_mapping.take(labels), columns=['size'])
    finalDf = pd.concat([principalDf, labels], axis=1)
    unique_labels = labels['size'].unique()

    for label in unique_labels :
        temp_dataset = finalDf[finalDf['size'] == label]
        label_rows = []
        for idx,row in temp_dataset.iterrows() :
            label_rows.append(row.values.tolist())
        projection_dataset[label] = label_rows
else :
    clusters = pd.DataFrame(clusters, columns=['cluster'])
    finalDf = pd.concat([principalDf, clusters], axis=1)
    for cluster in range(0,4) :
        temp_dataset = finalDf[finalDf['cluster']==cluster] #.drop(columns=['cluster'])
        label_rows = []
        for idx,row in temp_dataset.iterrows() :
            label_rows.append(row.values.tolist())
        projection_dataset["Cluster_"+str(cluster)] = label_rows
```

Below are the 2 scatter plots for randomized and stratified datasets:



Top-2 PCA scatterplot for randomized data



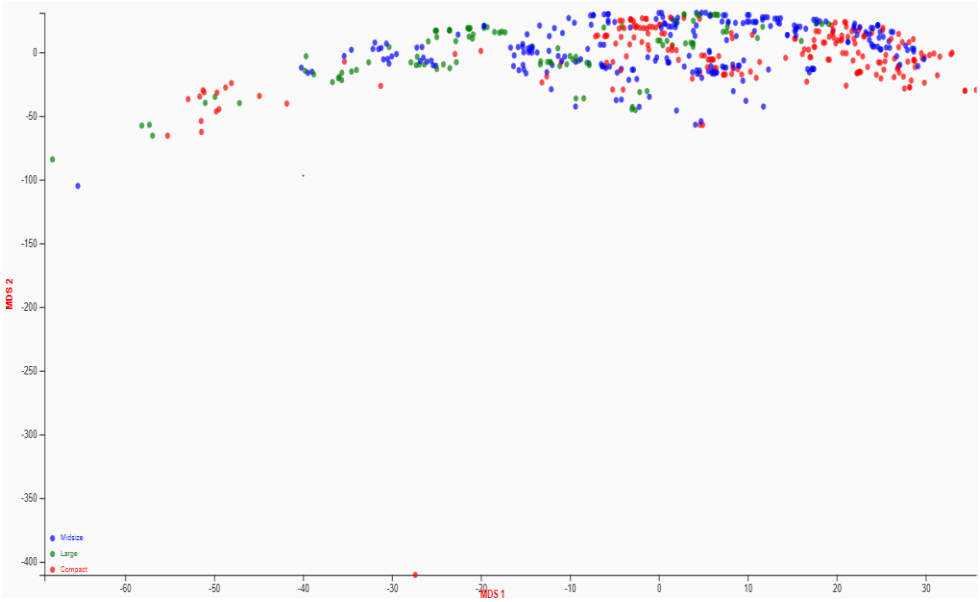
Top-2 PCA scatterplot for stratified data

In the randomized data we can clearly see datapoints cluster based on the style of the car, which would make sense and suv,sedan, etc would cluster together. However, in the case of stratified data the clusters yielded less information.

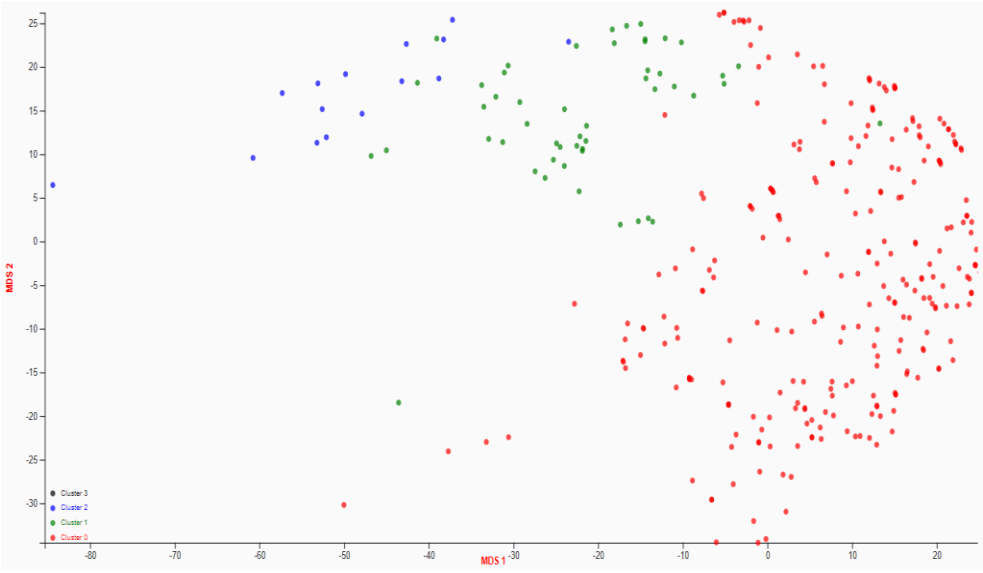
In the second step I was asked to find the MDS correlation between the datapoints, this was done with the sklearn.manifold MDS module. Note that the distances between datapoints was precomputed either Euclidean for dissimilarity or Correlation for the similarity matrix. The code snippet below demonstrates how this was done :

```
scaler = StandardScaler(with_mean=True,with_std=True) #
x_train = scaler.fit_transform(dataframe)
dataframe = pd.DataFrame(x_train,columns=dataframe.columns)
for distance in [distance_metric] : #
    matrix = metrics.pairwise_distances(dataframe,metric=distance)
    mds = MDS(n_components=2,metric='precomputed')
    mds_data_transformed = mds.fit_transform(matrix).tolist()
    mds_data_transformed = pd.DataFrame(mds_data_transformed, columns=['MDS1', 'MDS2'])


    if data_type in ['completedata','randomizeddata'] :
        finalDf = pd.concat([mds_data_transformed, labels], axis=1)
    else :
        finalDf = pd.concat([mds_data_transformed, clusters], axis=1)

    mds_data[index][distance] = finalDf.values.tolist()
```

The diagram below shows the MDS computed using the dissimilarity matrix for the randomized & stratified dataset:

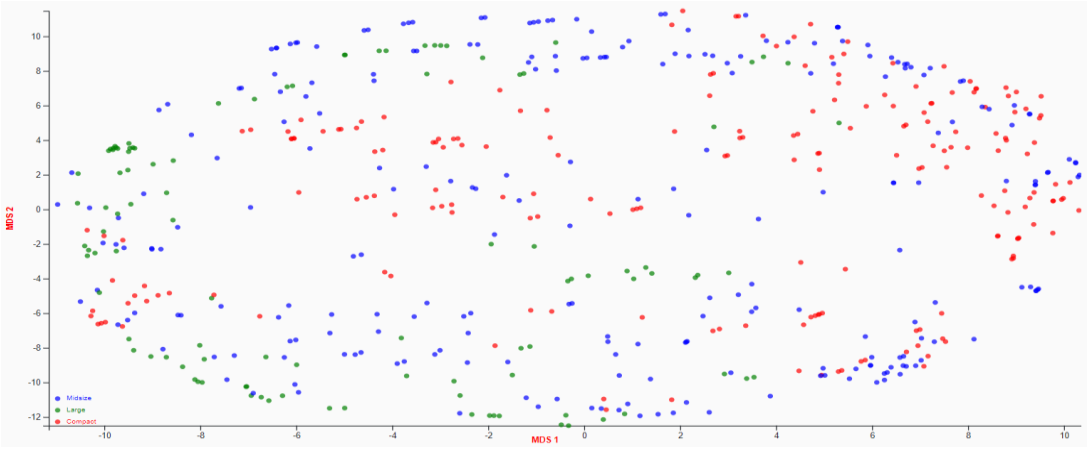


MDS Euclidean distance scatterplot (Randomized data)

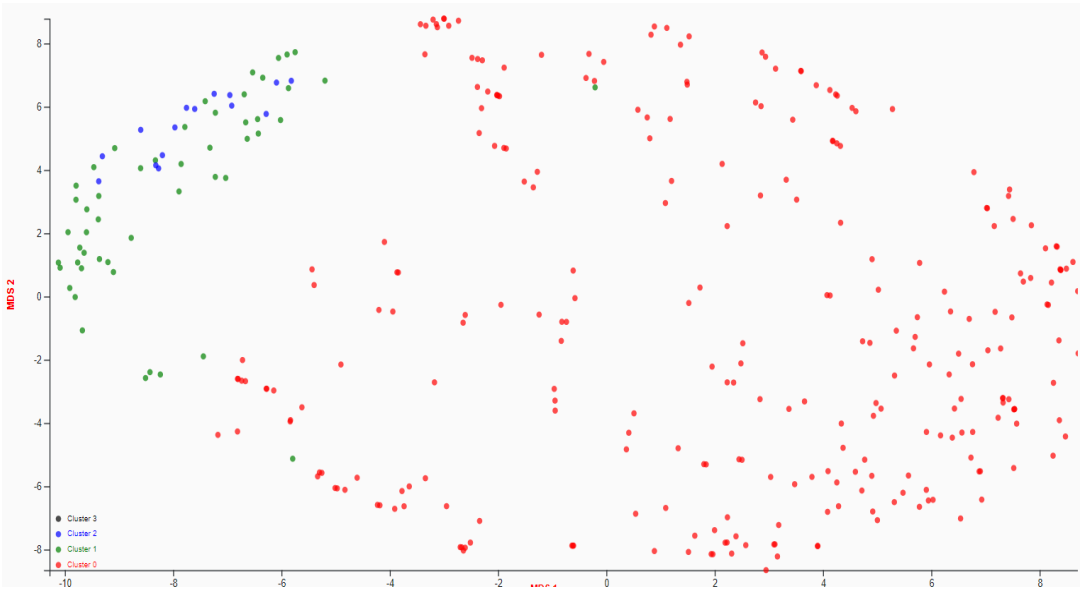


MDS Euclidean distance scatterplot (Stratified data)

Similarly, the MDS for randomized and stratified data was performed with correlation distance (essentially a similarity matrix). The diagrams below show the respective plots :



MDS Correlation distance scatterplot (Randomized data)



MDS Correlation distance scatterplot (Stratified data)

MDS with Euclidean distancing for randomized data produced clusters that were very close and did not yirld useful information, in the case of stratified data however which was separated using Kmeans we can indeed notice that datapoints of similar cluster indeed form groups.

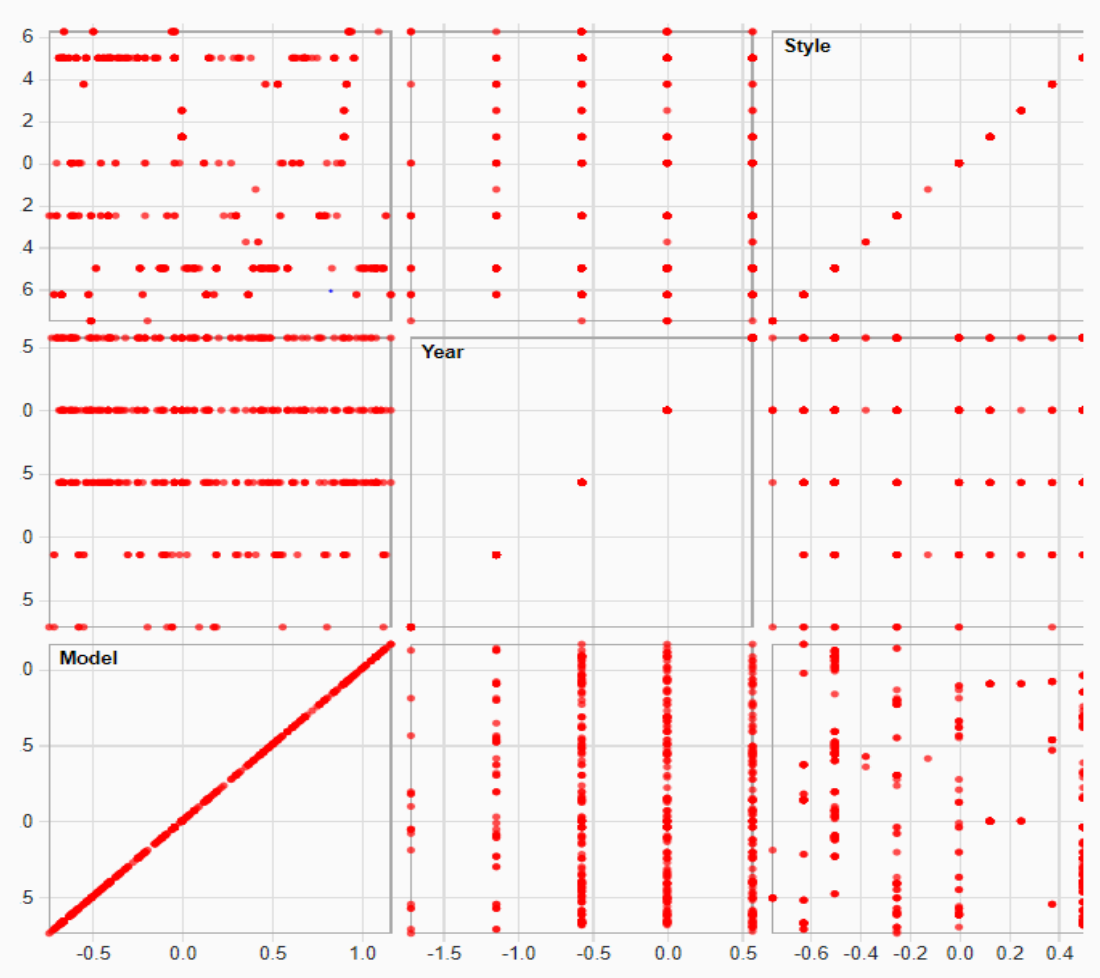
In case of MDS with Correlation we can see distinct clusters being formed, each cluster defining the group they belong to i.e car size in case of randomized data or cluster in case of stratified data. It appears as though datapoints that are similar to each other appear close to each other

The final subtask of task 3 was to perform a scatter plot matrix of the top 3 PCA loaded attributes, i.e perform a scatterplot of the top 3 significant attributes we found in task 2.

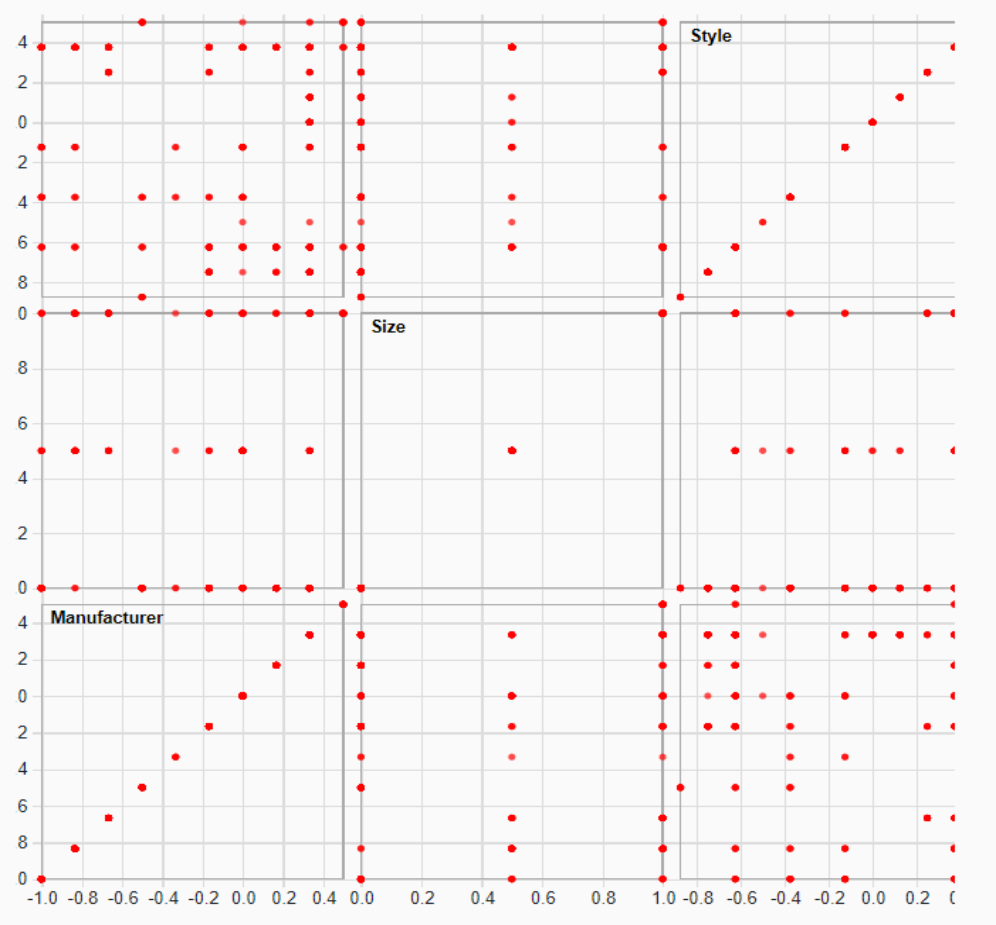
The code snippet for generating the dataset is as follows, note that the fetch top_3_attr function merely fetches the top 3 significant attributes :

```
significance = fetch_top_three_attr(data_type)
top_three = [ row['x'] for row in significance ][0:3]
dataframe = data_loader.data[data_type].loc[:,top_three]

scaler = RobustScaler()
x_train = scaler.fit_transform(dataframe)
dataframe = pd.DataFrame(x_train,columns=top_three)
plot_matrix = {
}
plot_matrix = []
for idx,row in dataframe.iterrows() :
    plot_matrix.append(row.to_json())

return jsonify(plot_matrix)
```

For this task I simply plotted each value of the 3 attributes against each other in a scatterplot, below is the output of the diagrams :



Scatterplot matrix of the top 3 PCA loaded attributes (Randomized data)



Scatterplot matrix of the top 3 PCA loaded attributes (Stratified data)

References

https://github.com/d3/

https://github.com/google/palette.js/tree/master

https://blog.risingstack.com/d3-js-tutorial-bar-charts-with-javascript/

https://www.freecodecamp.org/news/how-to-create-your-first-bar-chart-with-d3-js-a0e8ea2df386/

https://observablehq.com/@d3/brushable-scatterplot-matrix

https://bl.ocks.org/Fil/6d9de24b31cb870fed2e6178a120b17d