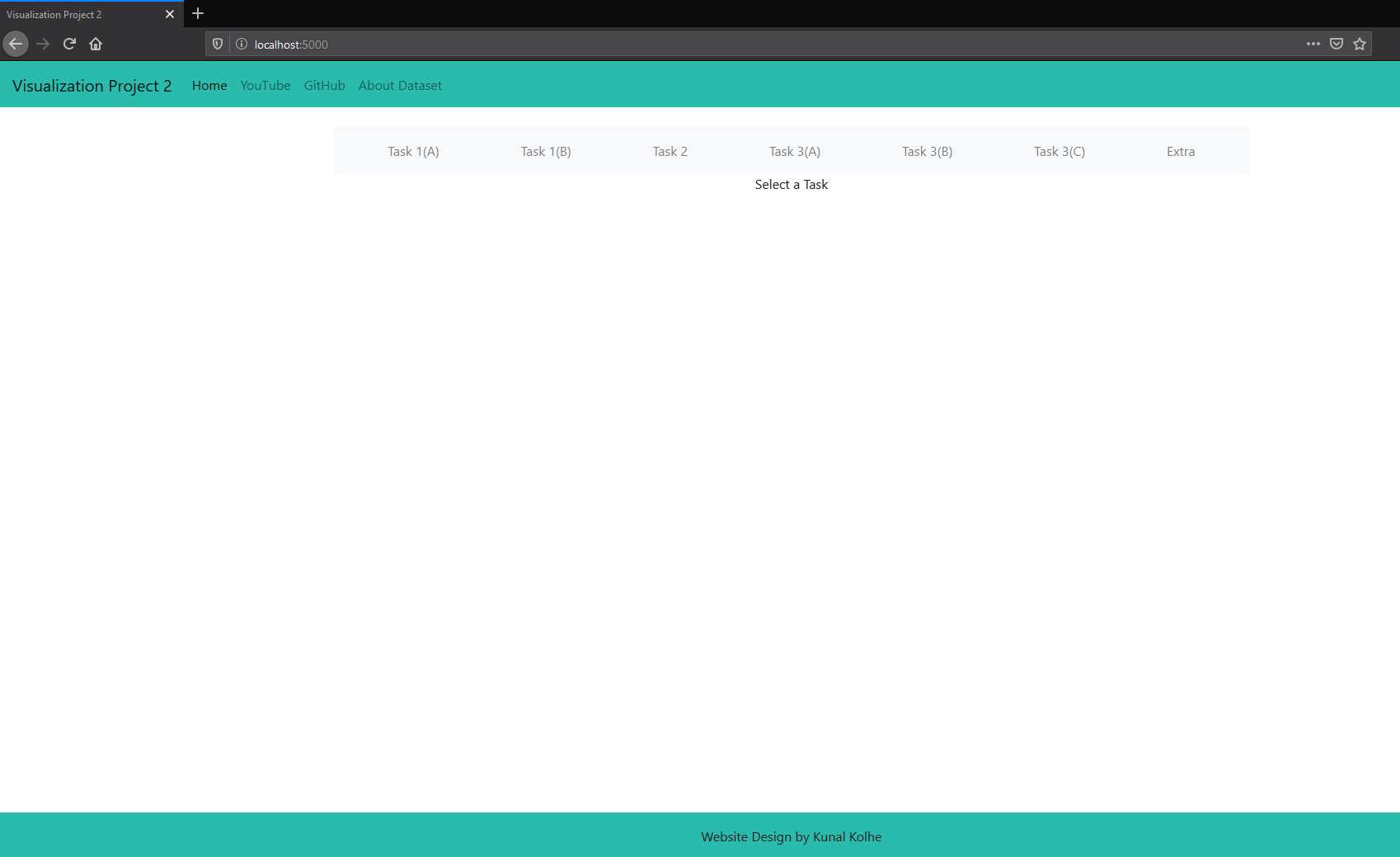
# Kunal Kolhe – 112748535

I used the same data as for the Project 1.

It is an Australian credit approval dataset consisting of 690 entries of 14 attributes and 1 target attribute. I got it from the UCI Machine Learning Repository. There are 6 numerical and 8 categorical attributes. The labels have been changed for the convenience of the statistical algorithms. For example, attribute 4 originally had 3 labels p,g,gg and these have been changed to labels 1,2,3.

## Flask:

I used flask web framework to deliver the files and used it as a server at the back end.



The UI for the project. The user will select the tasks and the respective function will run at the back end. The data is then served and rendered at the front end. The entire code is dynamic and the only file being read is the dataset. For each function, you will see different plots for the sampled data after refreshing due to the dynamic nature of the program.

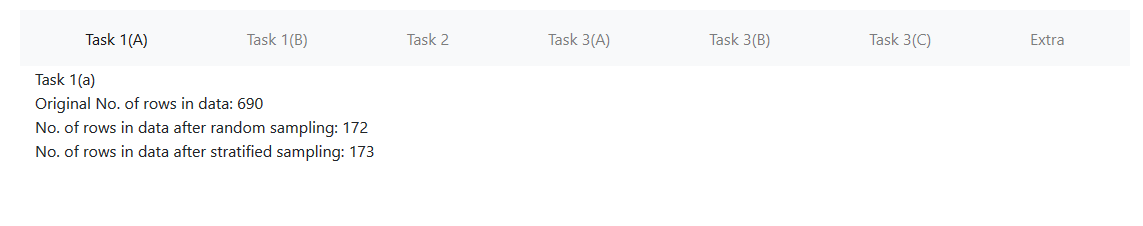
## Task 1(A):

Sampling:

Random Sampling: data.sample(frac=0.25) This code snippet just samples 25% of the data.

Stratified Sampling: sampledData=data.groupby('A15', group\_keys=False).apply(lambda x: x.sample(int(np.rint(0.25\*len(x))))).sample(frac=1).reset\_index(drop=True)

This code snippet groups data according to the target “A15” and then samples 25% of the data from each category.



Output for Task 1(A).

## Task 1(B):

Performing K-Means and optimizing k using elbow.

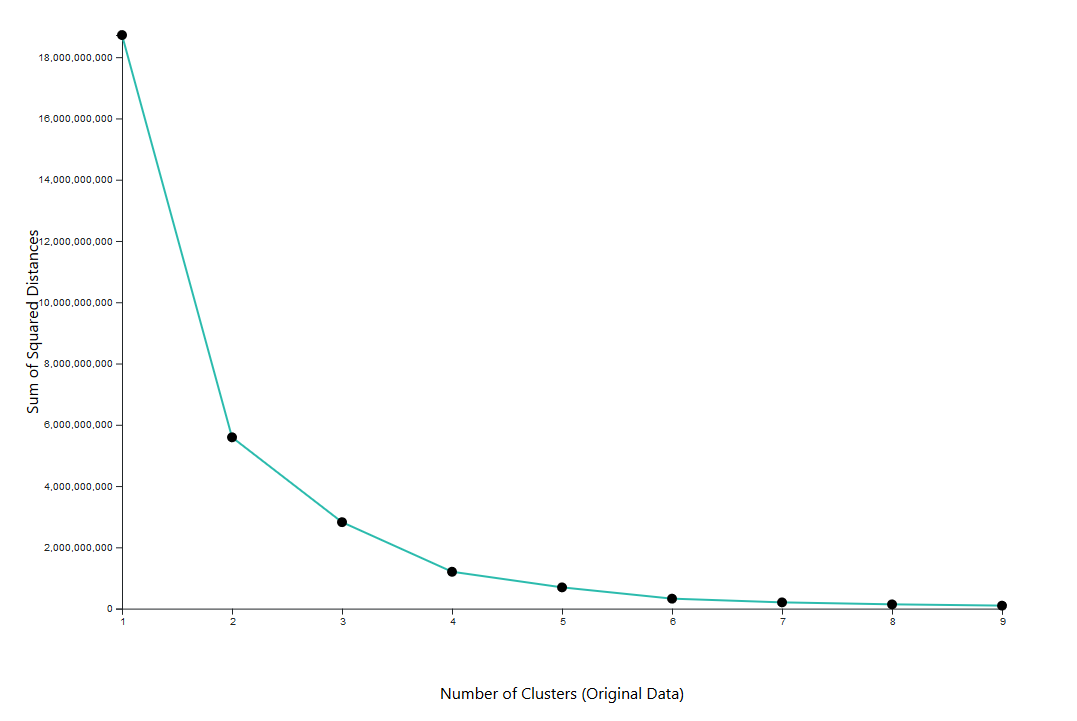
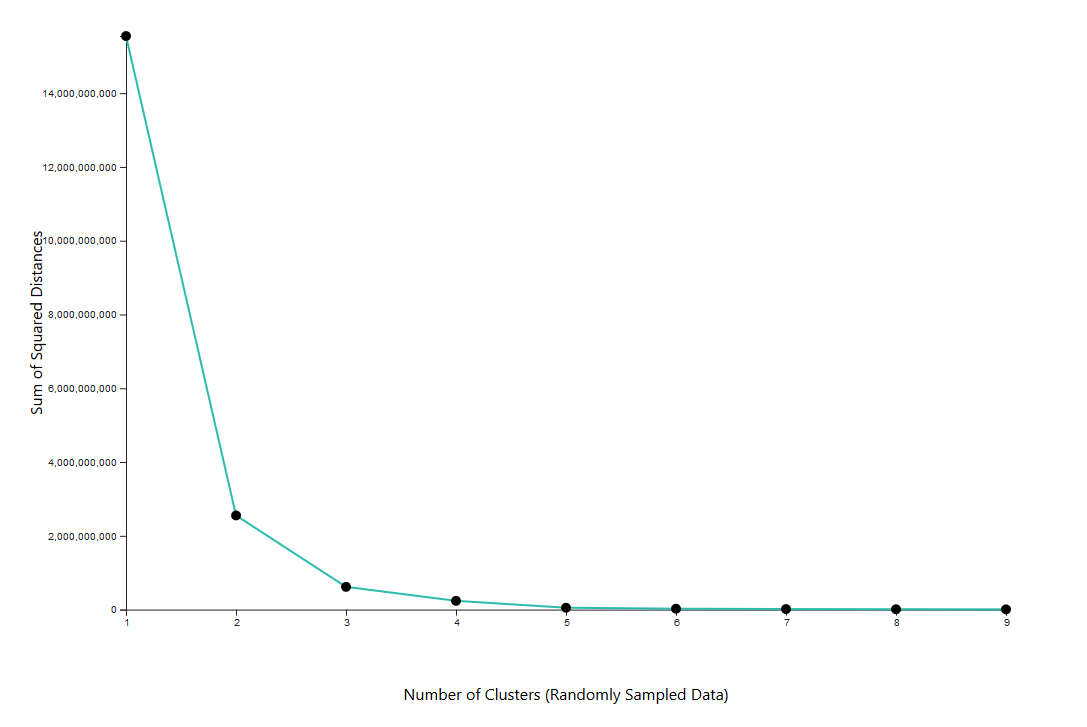
for i in range(1,10):

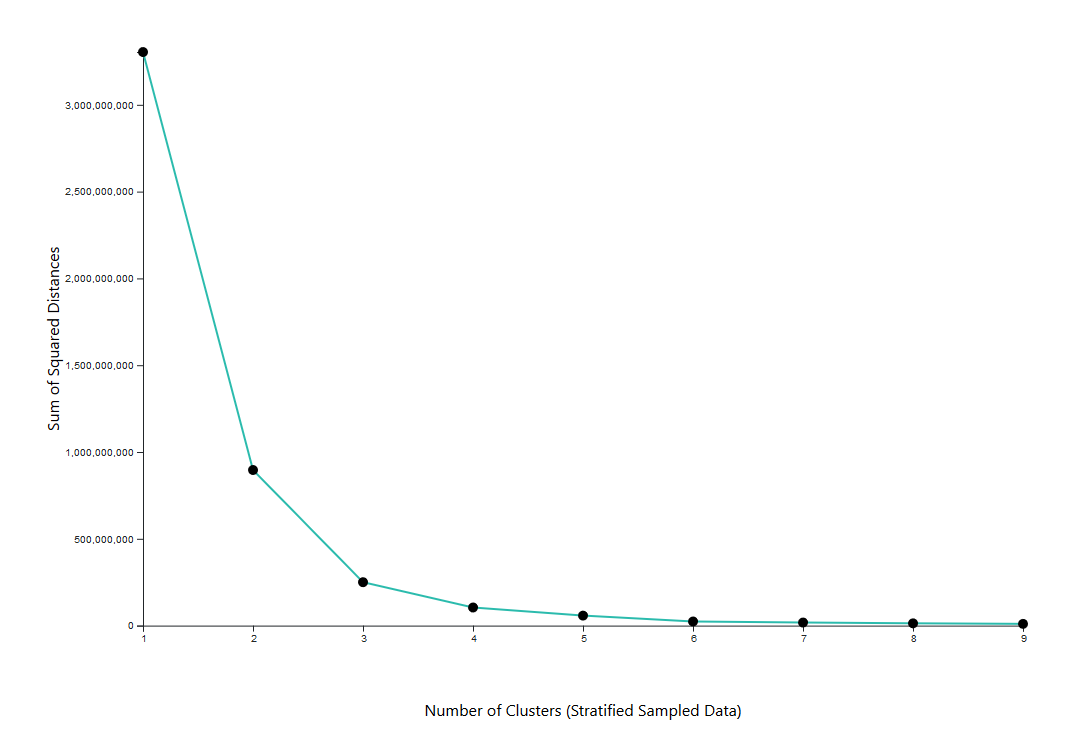
km = KMeans(n\_clusters=i)

alldistances = km.fit(data)

dictionary[i]=alldistances.inertia\_#totalDistance/i;

This code snippet calculates the sum of squared distances for 1 to 9 clusters. Then the data is sent to the front end, where the plots are displayed using d3.js

These plots show that the sum of squared distances decreases as k increases which is logical as clusters increase, less of the data will be farther away from the cluster centres.

Task 2:

Produce scree plots and mark the intrinsic dimensionality.

First I scaled the data, then calculated its eigen vectors and eigen values. I then plotted the eigen values as a bar chart. Intrinsic dimensionality of the data is when 75% of the variance is captured by the largest eigen vectors.

A = np.asmatrix(x.T) \* np.asmatrix(x)

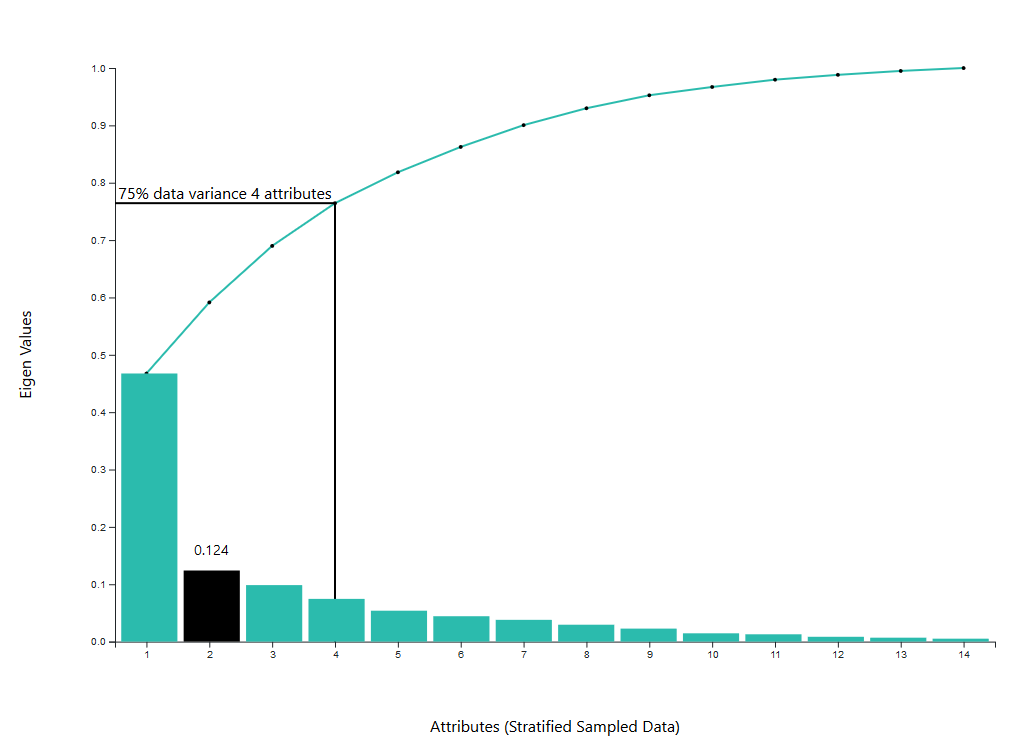
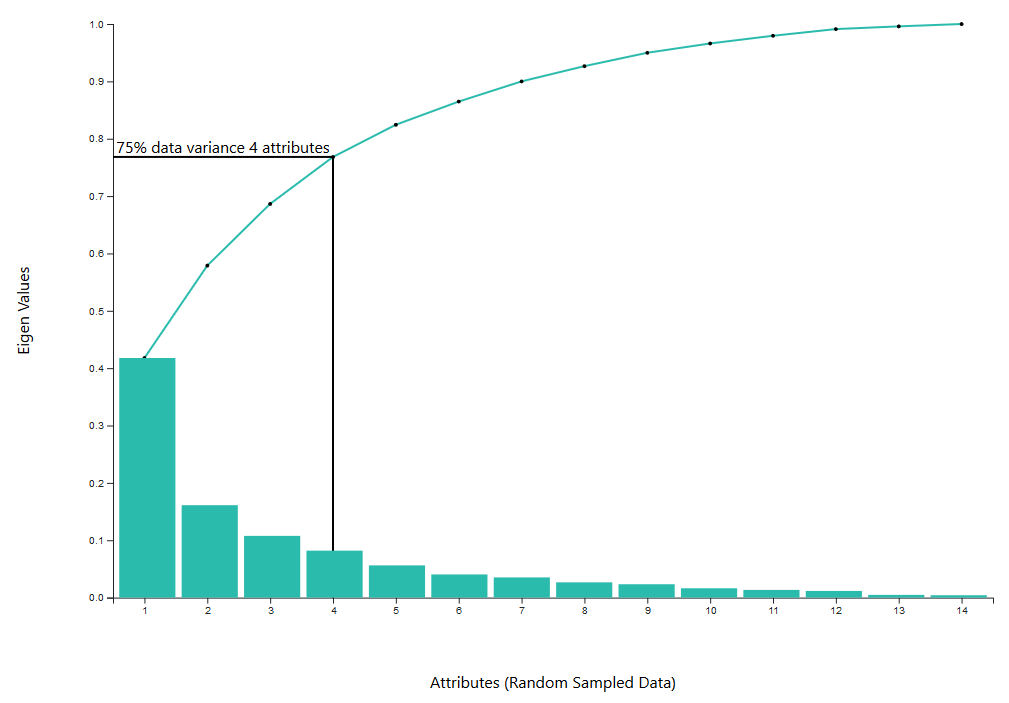
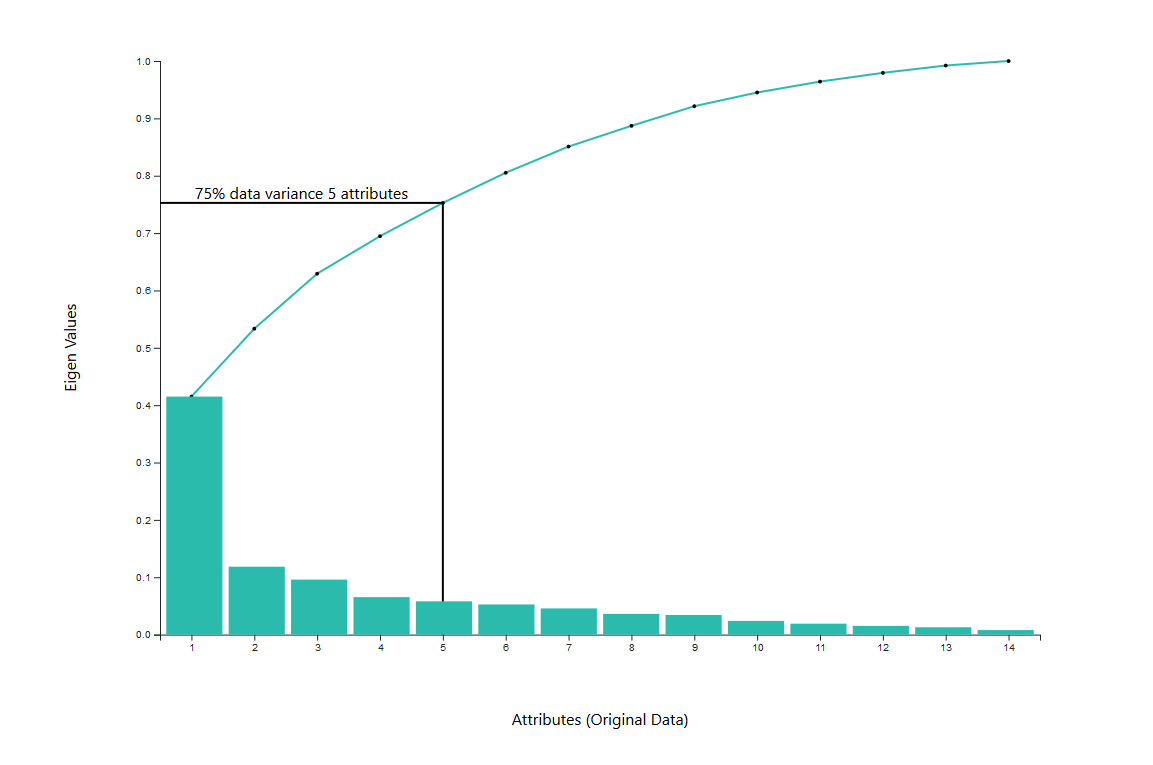
U, S, V = np.linalg.svd(A)

eigVals = S\*\*2 / np.sum(S\*\*2)

cumulative=[sum(eigVals[:i]) for i in range(1,15)]

In this code snippet, I am calculative the eigen values and then taking their cumulative in another array, for plotting.

The resultant Graphs:



As you can see 75% of the data variance is captured by 4 variables after sampling. But this depends on the sample and changes between 4 and 5. The bars have a tip to show the values. Almost always, the initial eigen values for the sampled data are higher than the unsampled data.

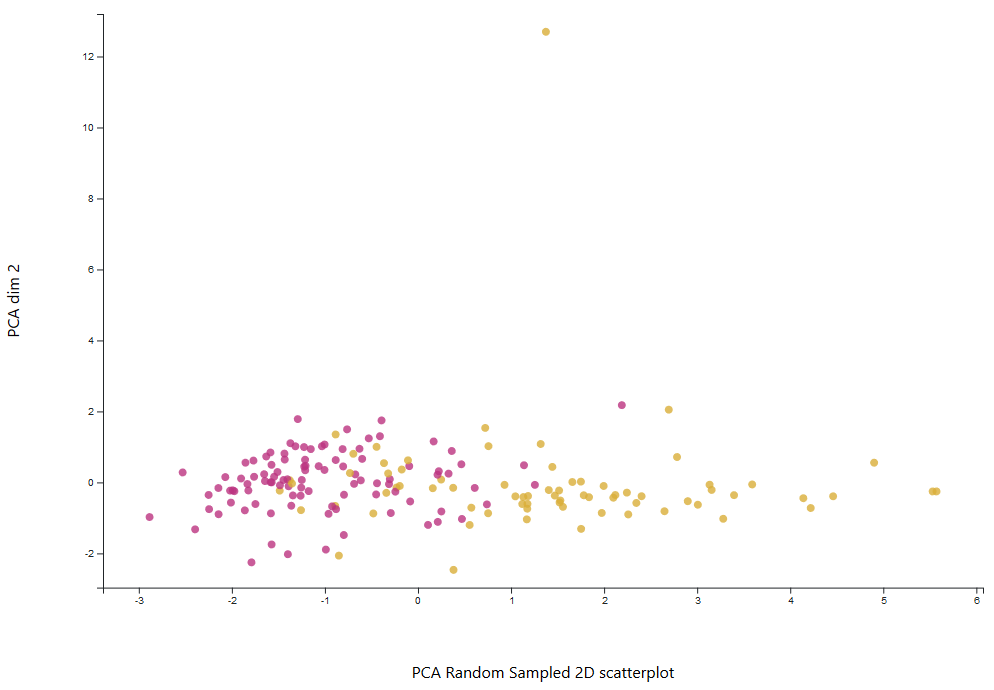
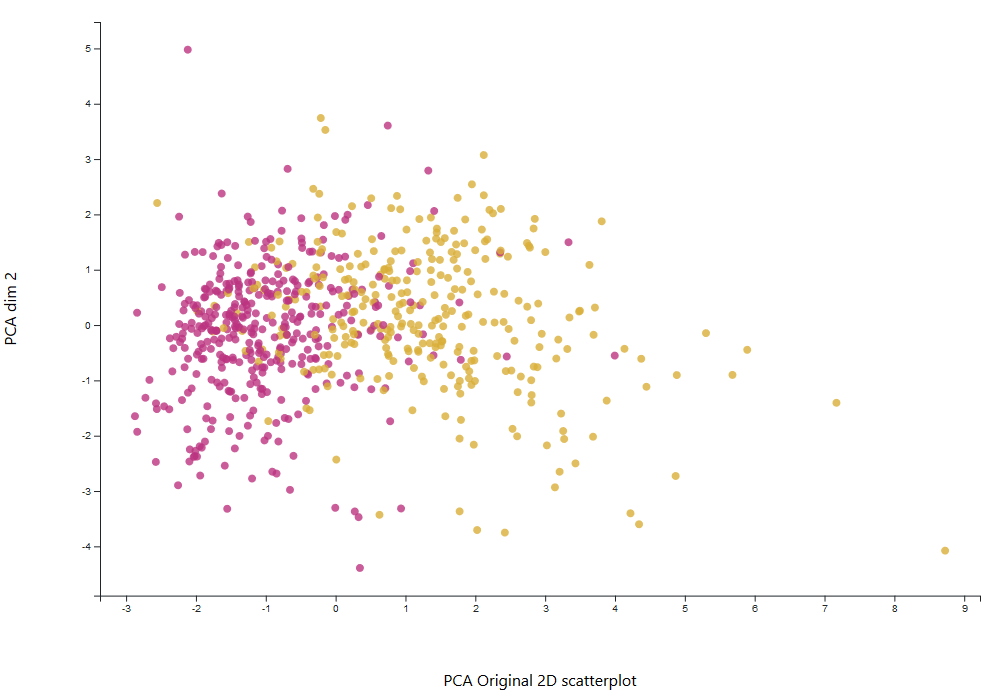
I got the 3 attributes with highest PCA loadings and used it for the next Task.

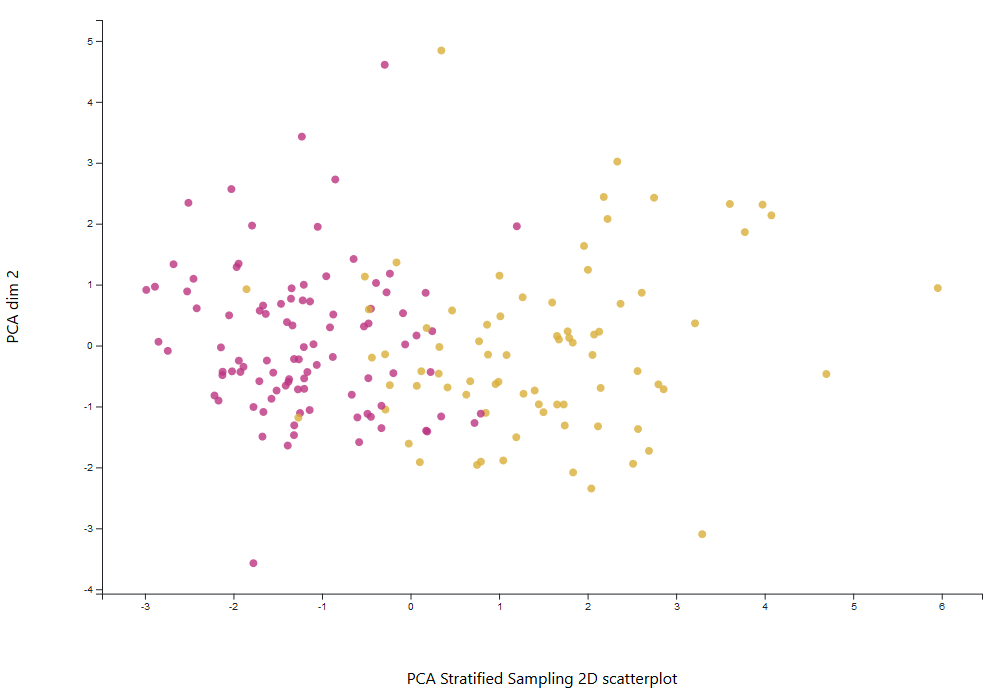
## Task 3:

pca = PCA(n\_components = 2)

originalPCA = pca.fit\_transform(dataOriginal)

This code snippet is used to get the 2 PCA loadings with the highest values in OriginalPCA. This array is plot as a 2D scatter plot color coded as per the target value for the respective point.





As you can see, when the data is broadcasted on 2 PCA vectors, we can see the separation between the data in 2D. Based on that, we can estimate that even if we use all the attributes to find a boundary, we will be successful with a high margin.

Based on the random sample, the scatter plot changes, but the apparent separation of “colors” remains.

These plots can be used as evidence that kmeans, logistic regression will work well on this data.

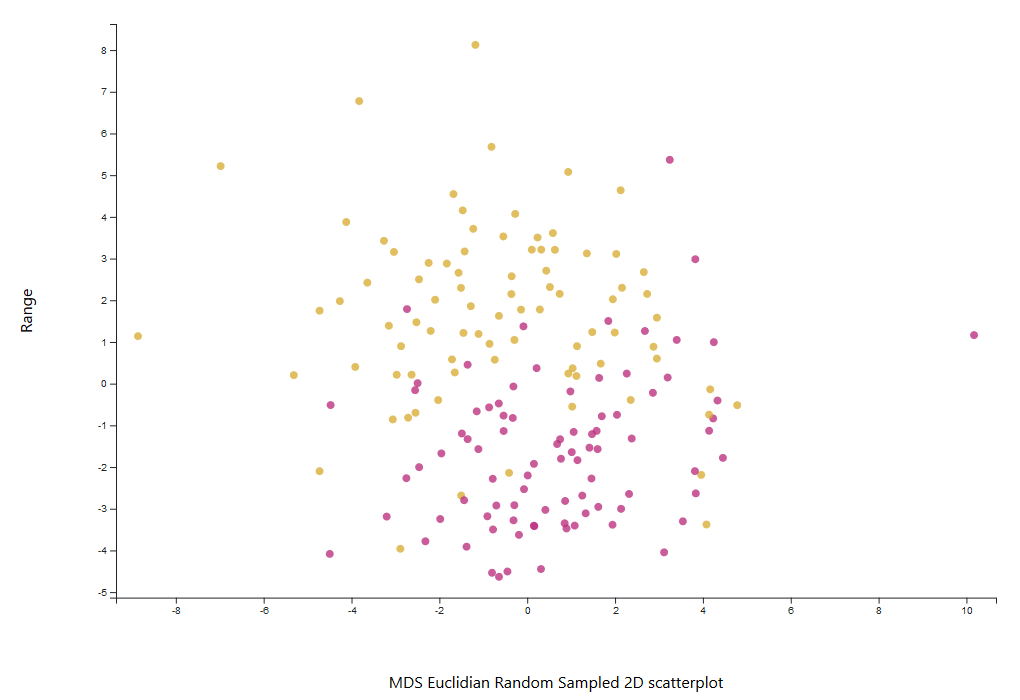
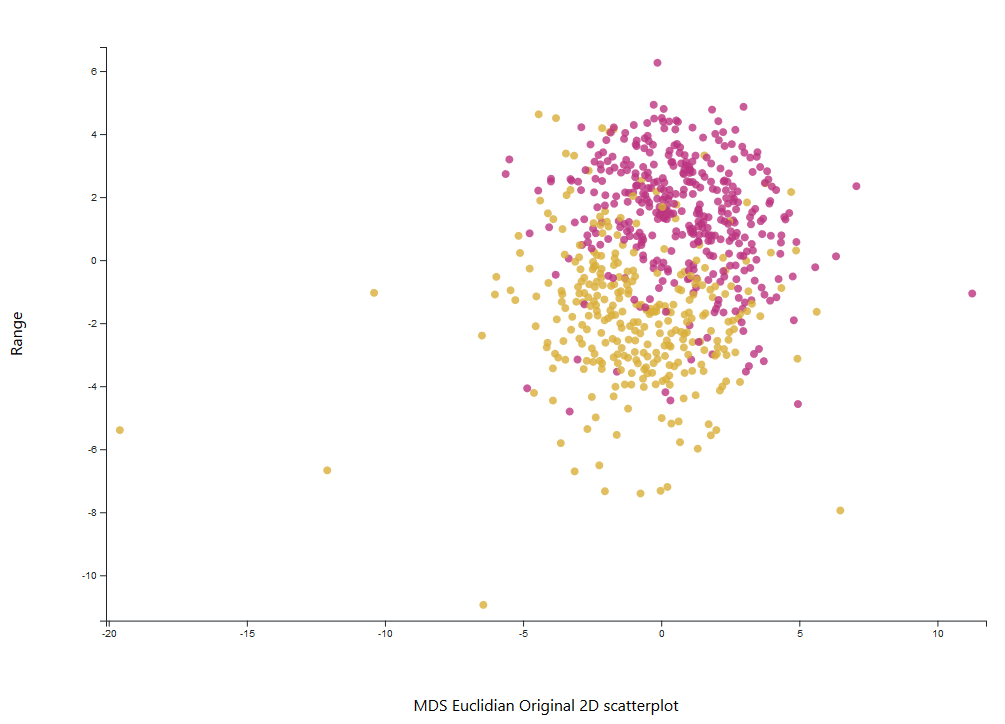
## Task 3(B):

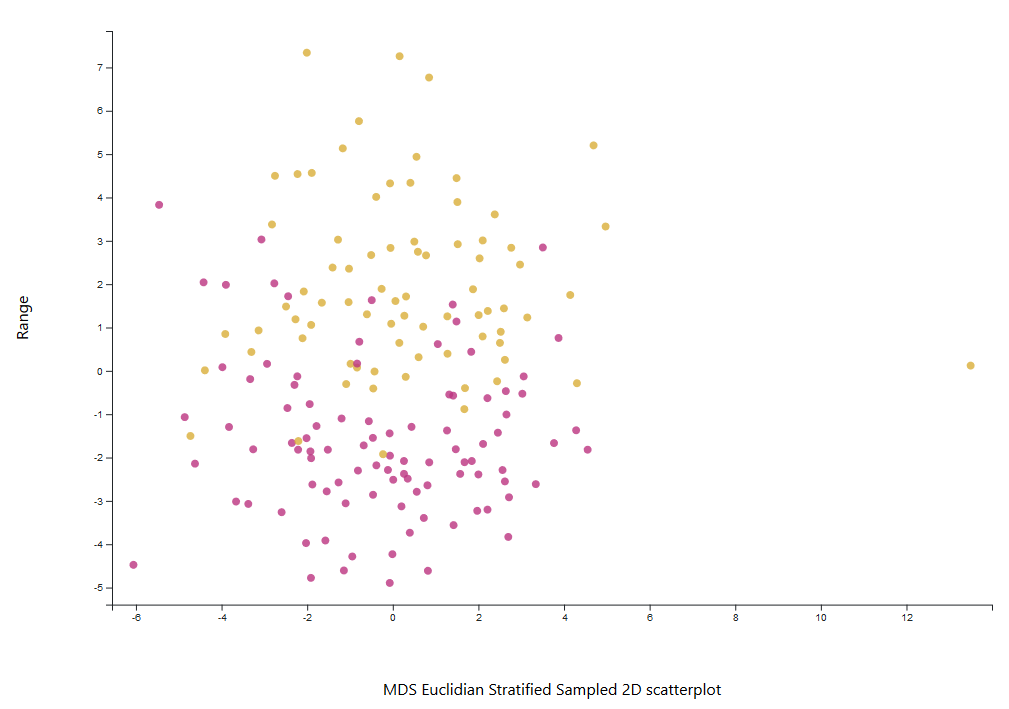
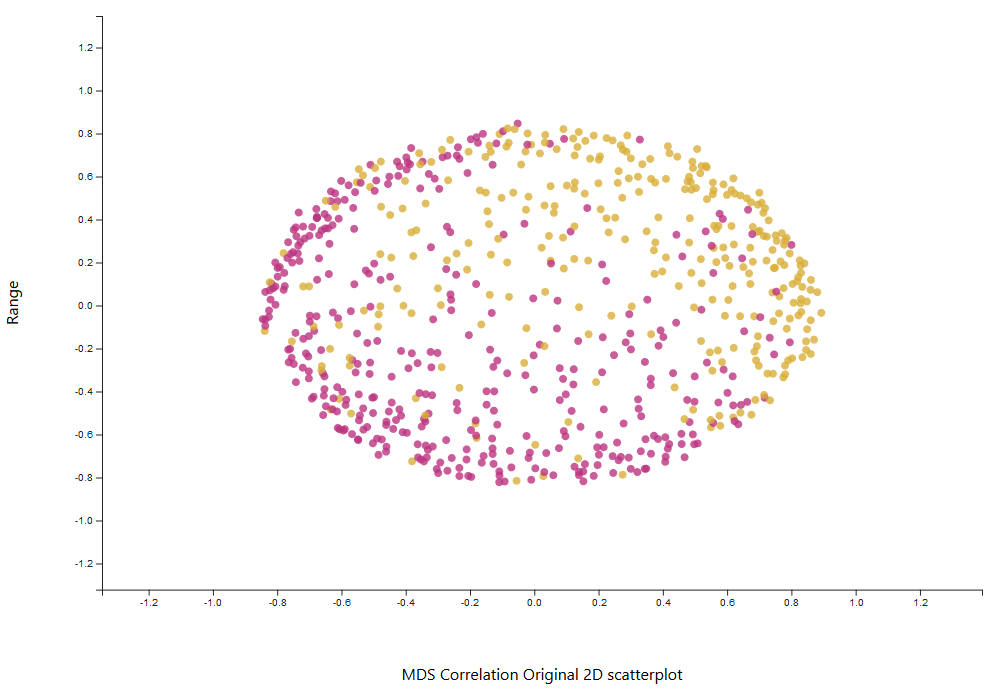
mds\_data = manifold.MDS(n\_components=2, dissimilarity='precomputed')

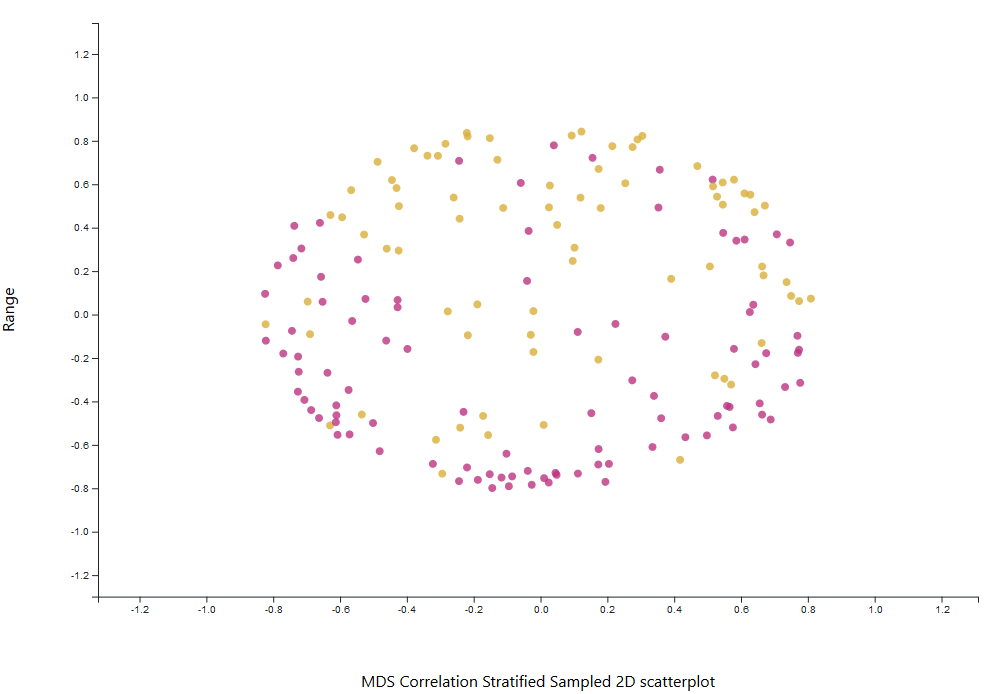
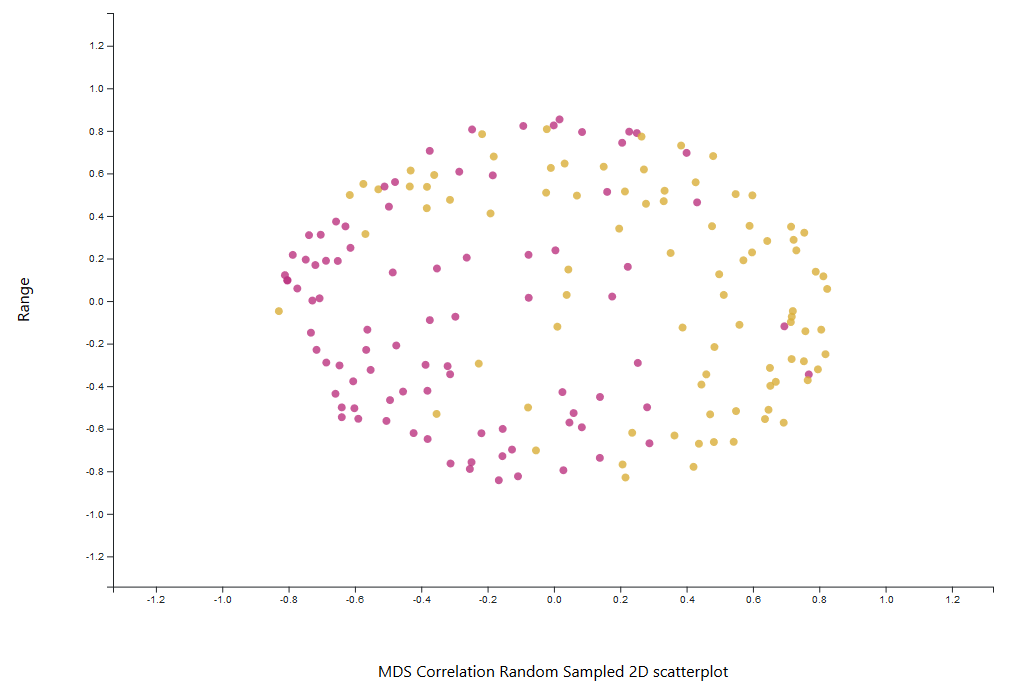
similarity = pairwise\_distances(dataOriginal, metric='euclidean')

originalMDSEu = mds\_data.fit\_transform(similarity)

This code snippet shows how multidimensional scaling can be done using sklearn. Since we want to plot on a 2D scatterplot, the number of components is 2. In the second line, the metric is Euclidean which is changed to Correlation when we want the distance metric to be correlation. The data must be scaled before being fed to this otherwise, result will not be proper. These plots are then displayed using 2D scatterplot.



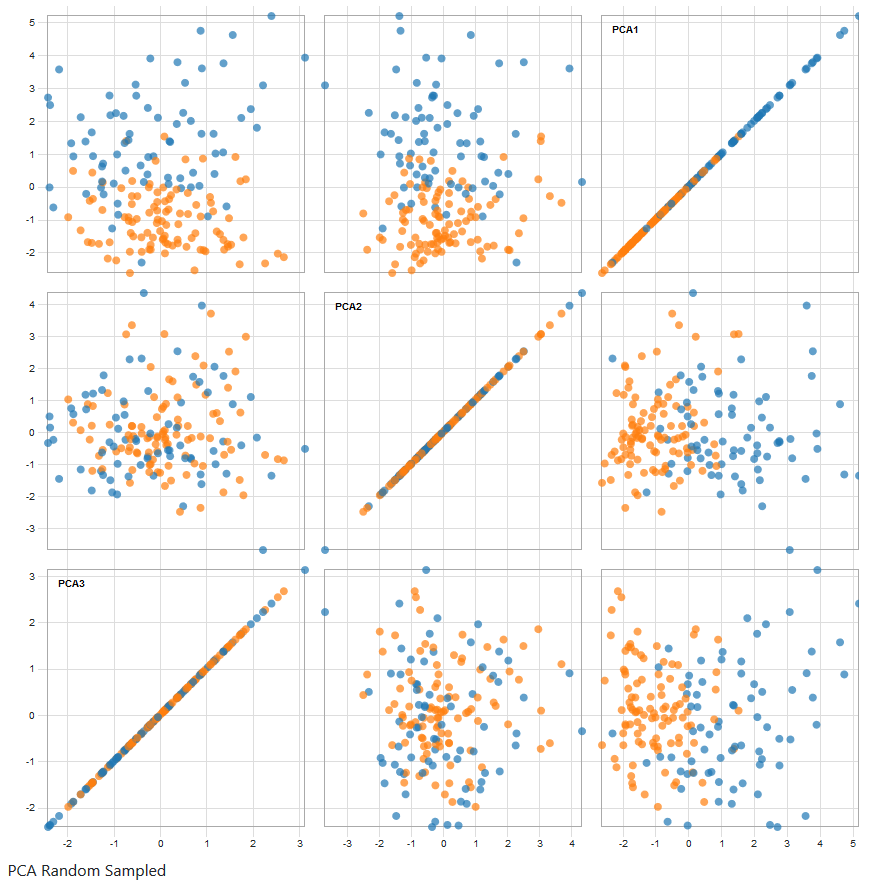
 



Multi Dimensional Scaling Plots using Euclidean Distance and Correlation Distance metrics. We can see the separation between the target classes in these plots too.

## Task 3(C):

The 3 highest PCA loadings have to be plotted into a scatterplot matrix.



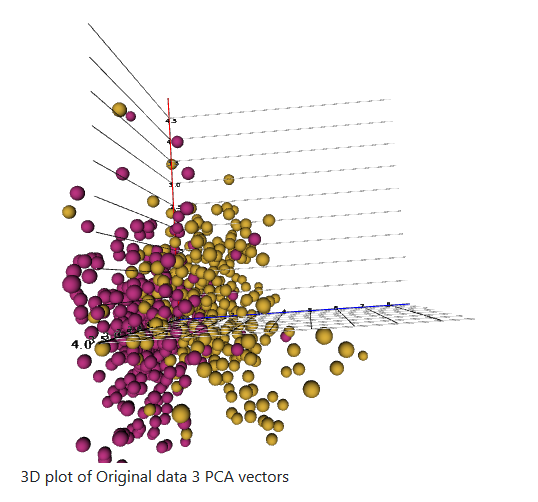
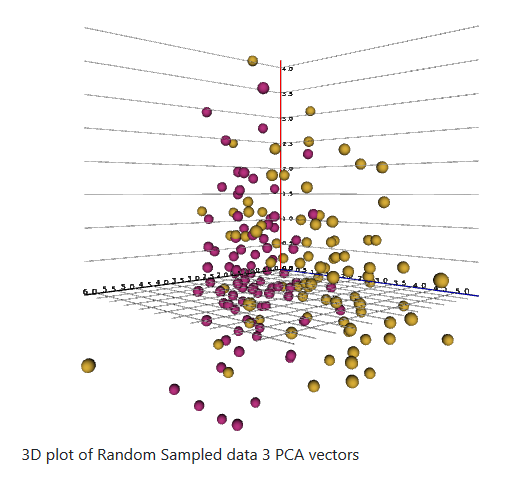
The 2D scatterplot matrix for the original Data, random Sampled Data and the stratified sampled Data.

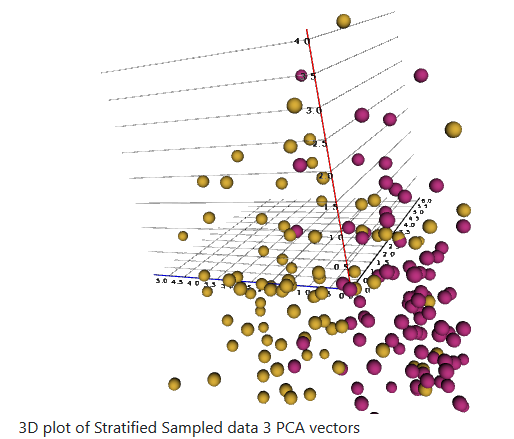
It has a select function which can show all the respective points in all the other plots.

Using that, we can see that in one of the dimension, the data is not being separated as well as the other dimensions. We can hence conclude that, for separation, we can get a reasonably good separation even if we take only 2 PCA vectors. This can be useful if the dataset is huge and applying algorithm for separation will be cost intensive. So in 2D, the cost of separating the data will be much less.

## Extra:

3D plots of the highest 3 PCA vectors.

I plotted the first 3 PCA vectors into a 3D scatter plot. The 3D matrix can be rotated and zoomed to view the data from all angles. The 3D plot also has mouse hover functionality. Value is displayed on mouse hover. The data points are again color coded so that we can see the difference between the classes.

Youtube Link:

Github Link: