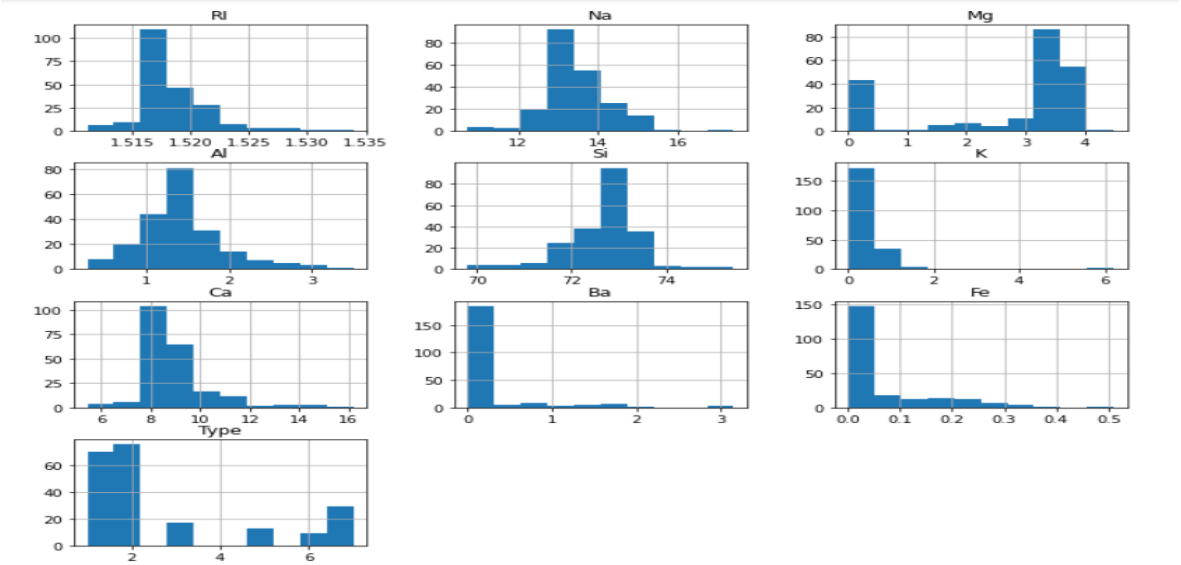
Pattern Recognition and Machine Learning
Lab 6 Assignment
KMeans, DBSCAN, KNN

Jatin Lohar
B21CS091

Question 1

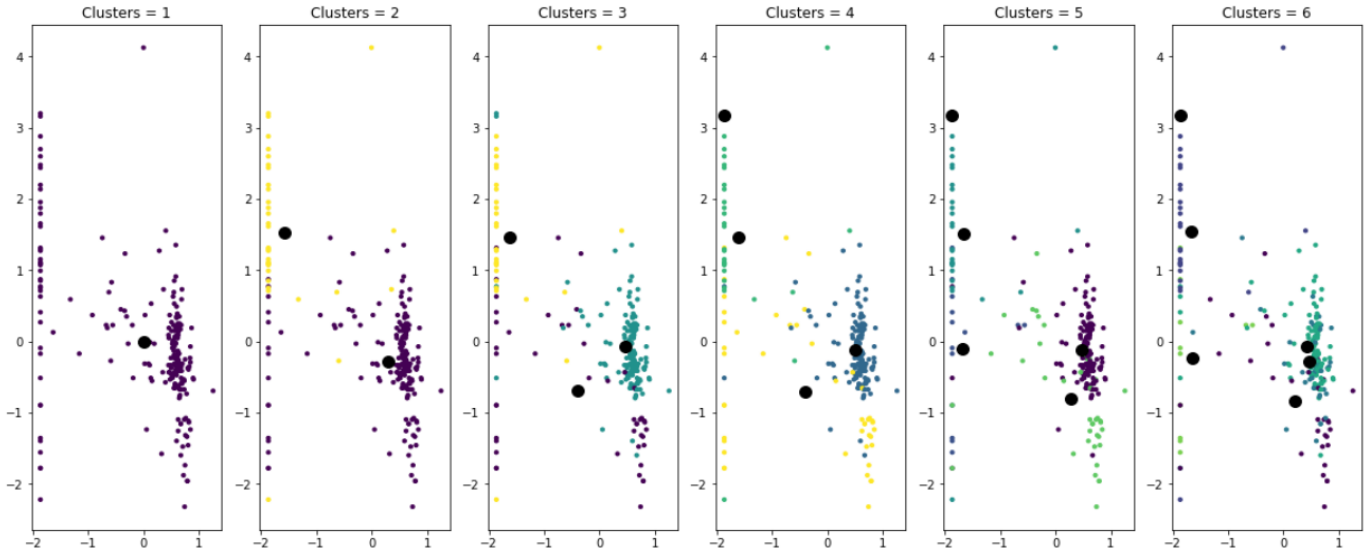
Imported ***'numpy' and 'pandas'.*** Then, using ***'pandas.read_csv'*** read the glass dataset. Then I checked for null values but did not find any. Plotted the histograms of the dataset.



Then pre-processed the data using ***'StandardScaler'***. Fitted and transformed the dataset using it.

Part a

Imported ***'KMeans'*** using ***'sklearn.cluster'***. Then plotted the graphs for different number of clusters by fitting the using different Kmeans with different ***'n_clusters'.*** Plotted the scatter plots for the same by using 2nd and 3rd feature (since it gave better graphs).



Part b

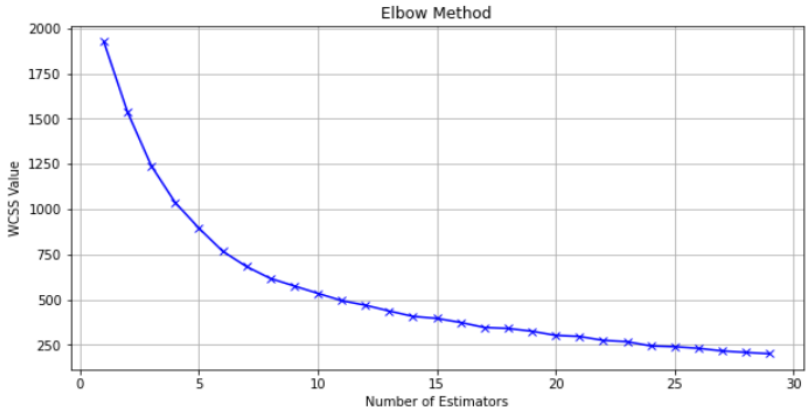
Imported '***silhouette_score'*** from ***'sklearn.metrics'.*** Then, calculated the scors with fitting for different Kmeans model with different ***'n_clusters'.***

```
Silhouette_score for 2 Clusters is -0.3698379448460016
Silhouette_score for 3 Clusters is -0.12236181813472817
Silhouette_score for 4 Clusters is -0.132854655589035
Silhouette_score for 5 Clusters is -0.1392812336996111
Silhouette_score for 6 Clusters is -0.2019891580604945
```

According to '***silhouette_score',*** optimal value of n_clusters should be 3. Since it had maximum value.

Part c

Used '***KMeans.inertia_'*** to find the WCSS values. Then plotted the graph to find the knee point.



The Knee-point according to the graph came out to be 7.

Part d

Imperted the ***'BaggingClassifier' and 'KNeighborsClassifier'.*** Then, for best results, in Bagging Classifer, used KNN with n_neighbors = 3. And for normal KNN, used n_neighbors = 4.

```
Accuracy with Bagging with KNN classifier : 0.8598130841121495
Accuracy (without Bagging) of KNN classifier : 0.8130841121495327
```

Found that BaggingClassifer, generally performed better than KNNClassifier.

Question 2

Part a, b

Used features of the KMeans class as ***'K', 'max_iterations', 'centroids', 'clusters'.*** While fitting, made centroids as the initialisers provided, else considered K random points in X and made them centroids. Then, made a assigned cluster as a empty 2D matrix, that will carry the indexes of the points. After the cluster points were assigned, then checked then calculated the mean of each cluster and checked whether it is same as earlier one or not. If yes, then this were the clusters, else ran the process again.

While predicting, we already had the clusters made. So made all the elements in a given cluster as the index of the cluster. Also, made a function to plot the centroids.

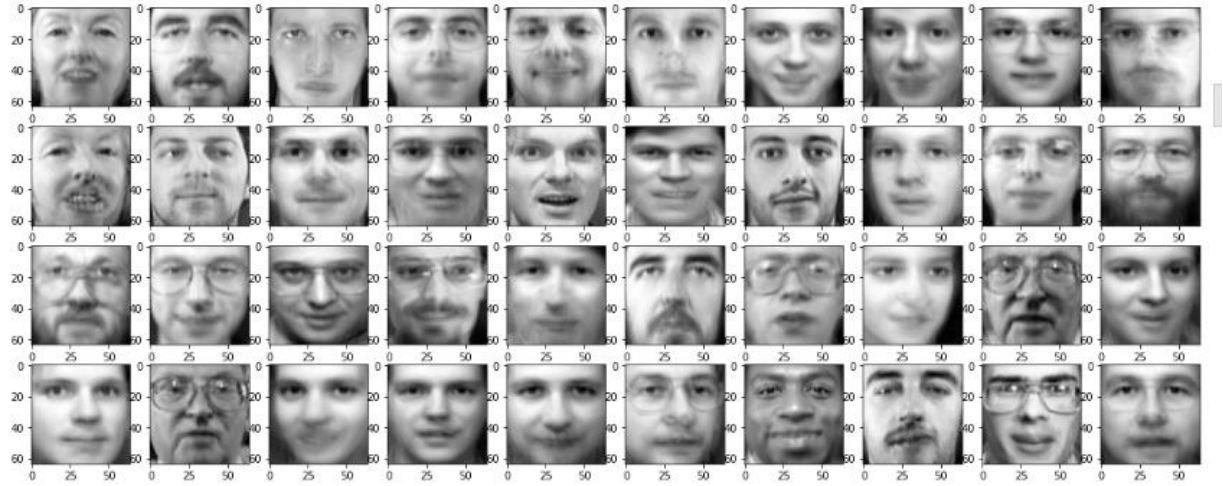
Part c

Imported the dataset. Made a list of some random number between 0 to 400 and the made them as the initialiser. Fitted and predicted the data, using the initialiser. Then printed the number of elements of each type.

```
0 -> 6
1 -> 6
2 -> 2
3 -> 9
4 -> 8
5 -> 14
6 -> 10
7 -> 14
8 -> 14
9 -> 12
10 -> 4
11 -> 1
12 -> 11
13 -> 18
14 -> 1
15 -> 10
16 -> 3
17 -> 12
...
```

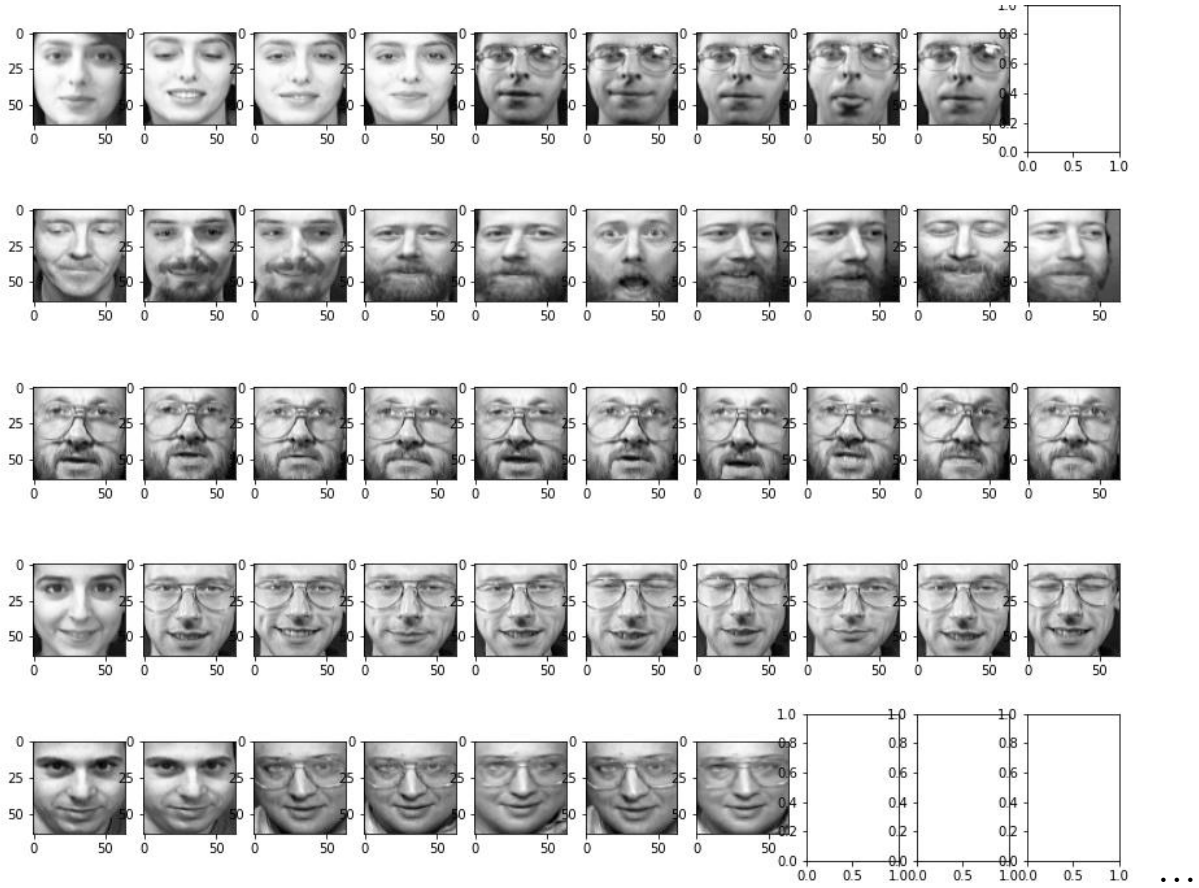
Part d

Used the function to plot the cluster centroids.



Part e

Plotted 10 indexes of each cluster. If the elements in a cluster were less than 10, then plotted the available ones. Found the classifier could cluster the similar/same faces.

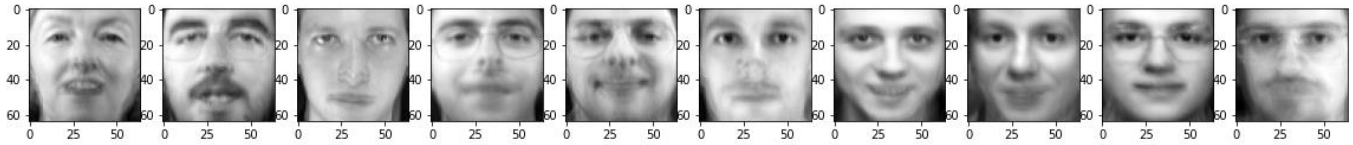


...

Part f

Trained the model again for k = 10 and 10 random initialisers. The found the number of elements in each cluster and also plotted the centroids.

```
0 -> 36
1 -> 40
2 -> 47
3 -> 46
4 -> 31
5 -> 31
6 -> 47
7 -> 37
8 -> 40
9 -> 45
```



Part g

Plotted 10 images of each cluster.

...

It could cluster pretty well.

Part h

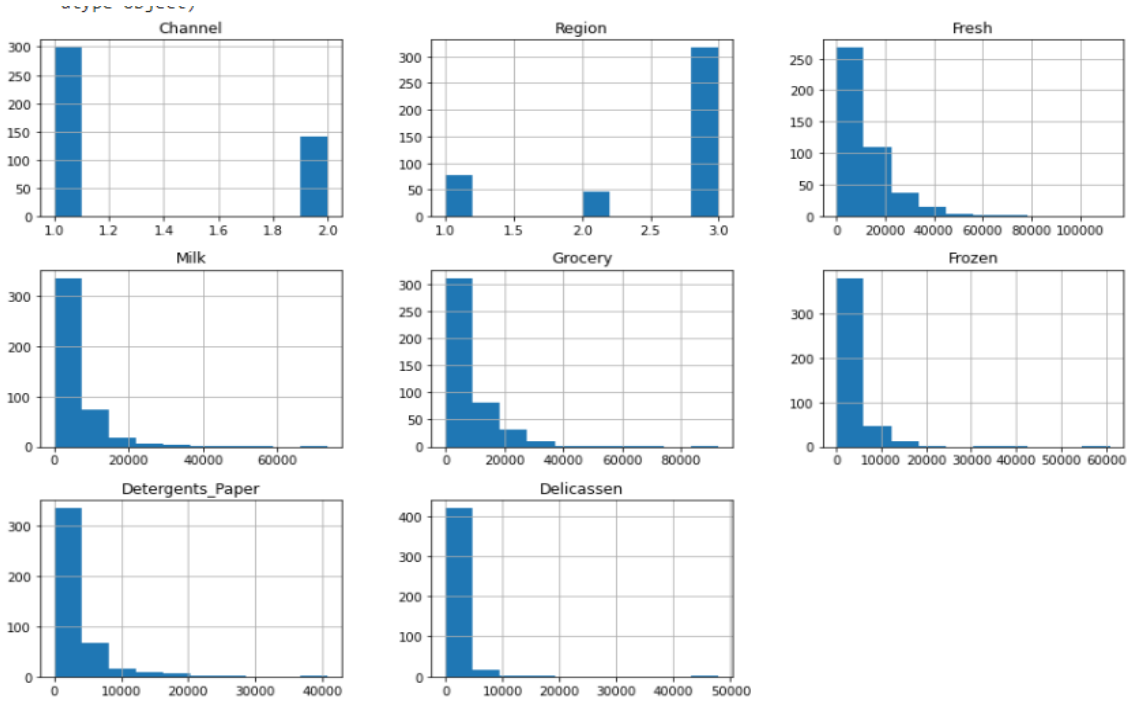
The Sum of Squared Error were as follows

```
The Sum of Squared Error for Part c is : 103324.0
The Sum of Squared Error for Part f is : 142805.0
```

So, Part c (40 Clusters) performed better than part f.

Question 3

Read the dataset using ***'pandas.read_csv'*** Found no null values in the dataset. Plotted the histograms.



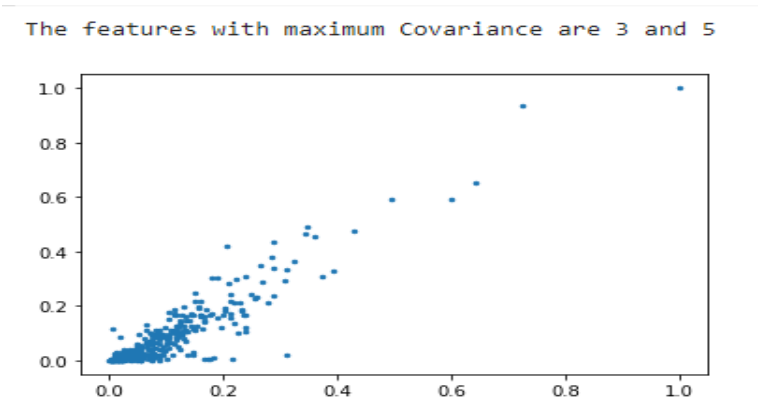
Part a

Used ***'MinMaxScaler'*** for converting every feature in a range of 0 to 1. Fitted and transformed the data. Also reduced the dimensions of the dataset so as to plot better scatter plots.

Part b

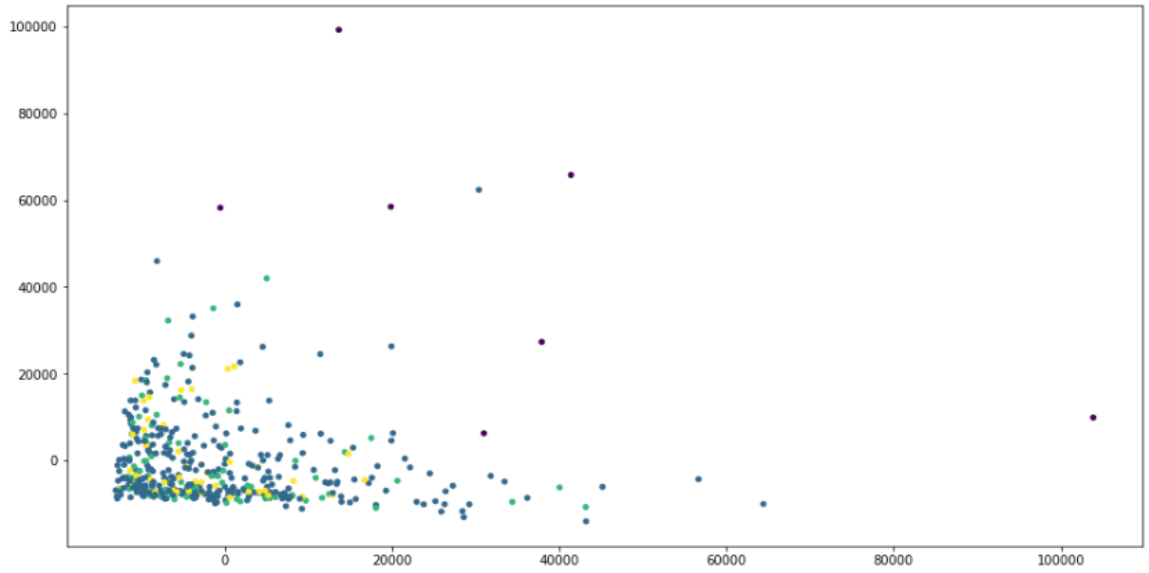
Found the covariance matrix. And the the features for which it was maximum.

Found that Feature-3 and Feature-5. Had maximum covariance. The plotted the scatter plot for the features.



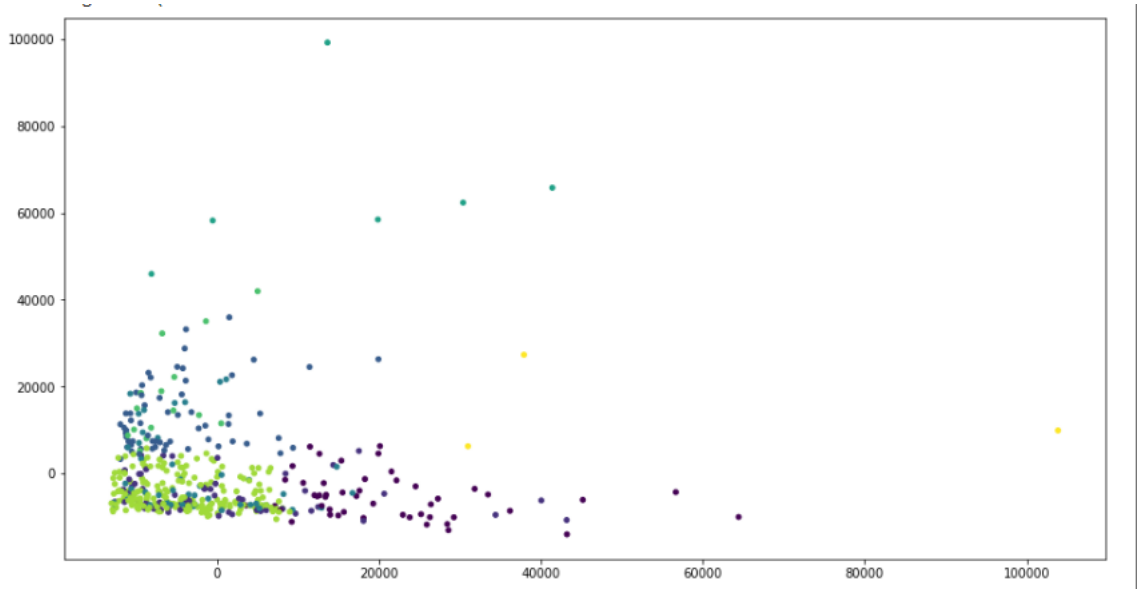
Part c

Imported ***'DBSCAN'*** from ***'sklearn.cluster'.*** Fitted the data and then found the predicted values using ***'DBS.labels_'***. Plotted the scatter plot for the same.



Part d

Fitted the data in KMeans model and then predicted the values.



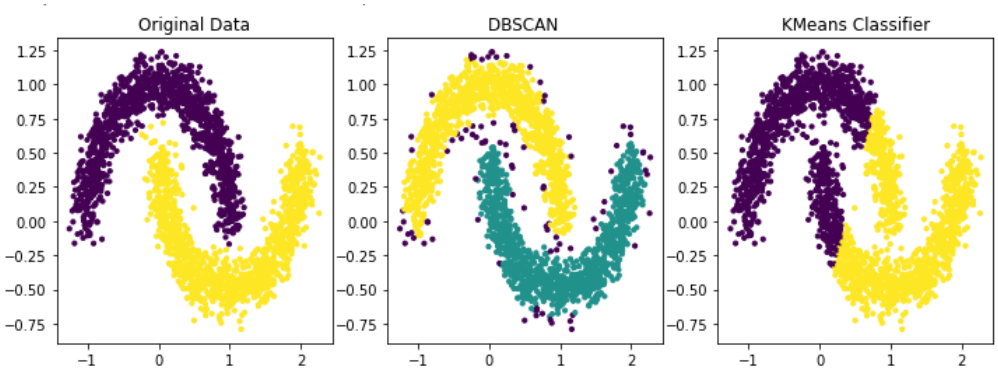
KMeans performed better clustering then DBSCAN as can be observed in the figure.

Part e

Made the dataset using ***'sklearn.datasets'.*** Added 400 random points in it so as to add noise.

For DBSCAN, used min_samples = 20, to get the best results.

In KMeans, since we have two classes, so used n_clusters = 2.



With given parameters, DBSCAN performed very well than KMeans.