Pattern Recognition & Machine Learning
Lab-6 Assignment

Name: Tanish Pagaria
Roll No.: B21AI040

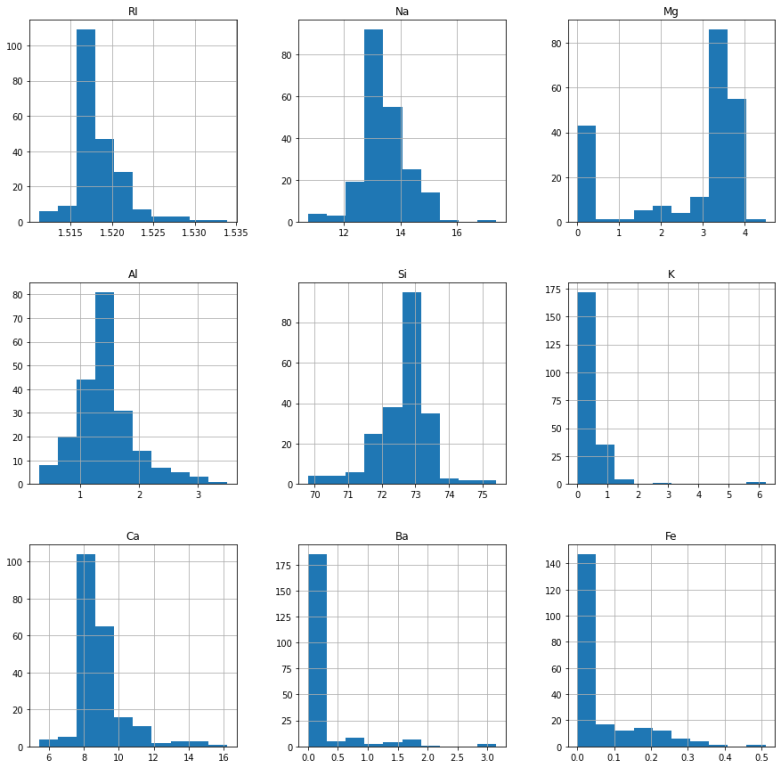
Question-1

Preprocessing and Analysis of the dataset

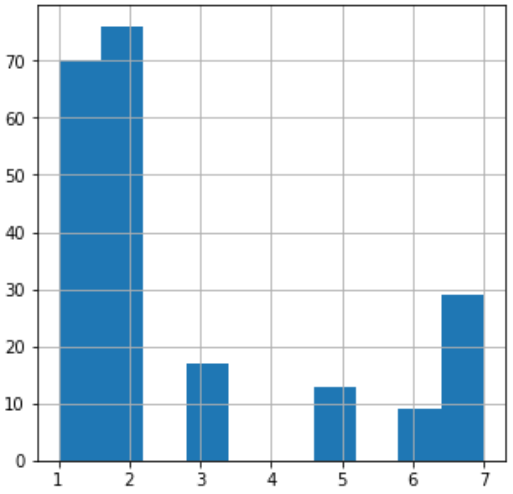
The dataset did not contain any NULL entries or NaN values. The dataset contained values in varied ranges for all the features, and hence, it was normalized.

```
Number of unique entries in RI : 178
Number of unique entries in Na : 142
Number of unique entries in Mg : 94
Number of unique entries in Al : 118
Number of unique entries in Si : 133
Number of unique entries in K : 65
Number of unique entries in Ca : 143
Number of unique entries in Ba : 34
Number of unique entries in Fe : 32
Number of unique entries in Type : 6
```

The number of unique entries corresponding to each feature/label



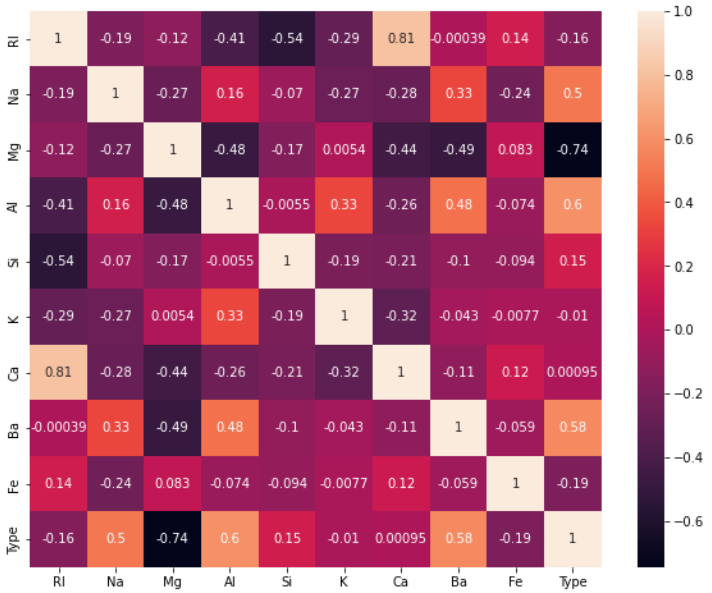
Features



Target

Histogram plots for the dataset

The correlation heatmap among the features, and also the label was plotted.

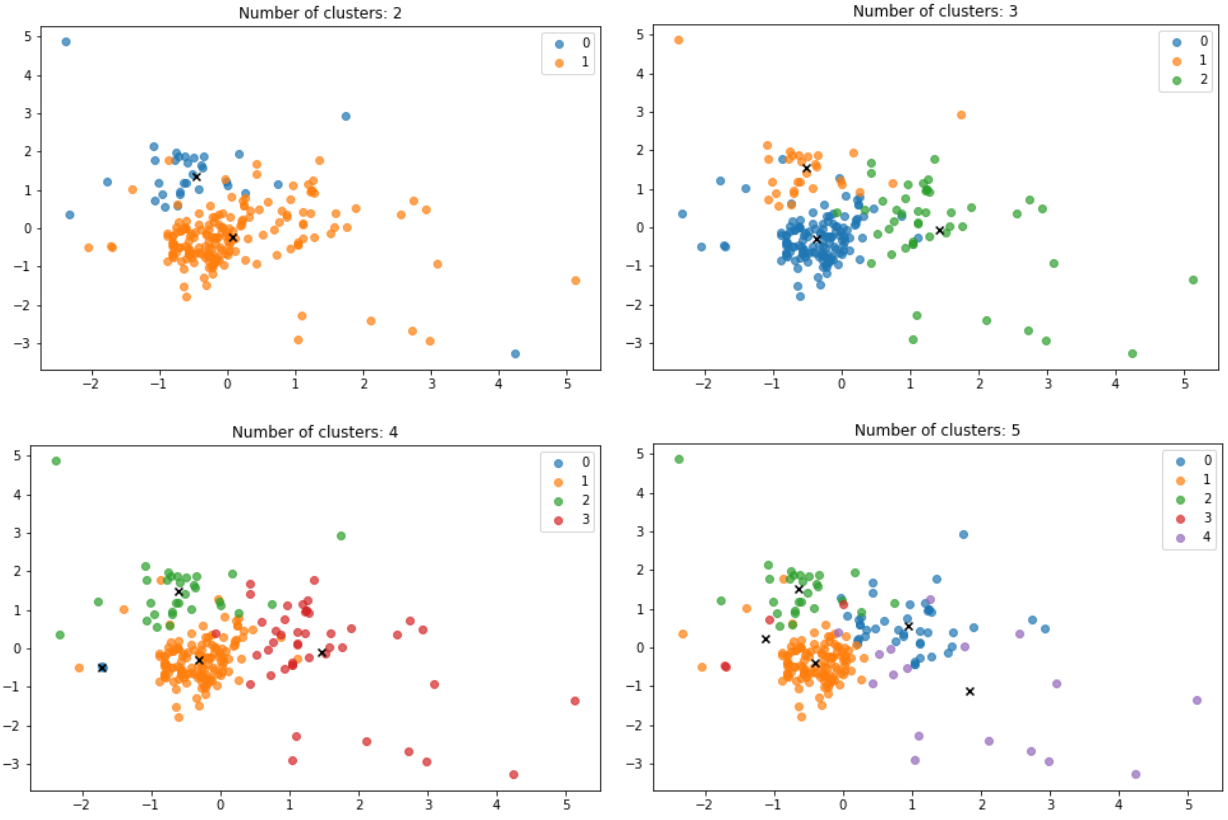


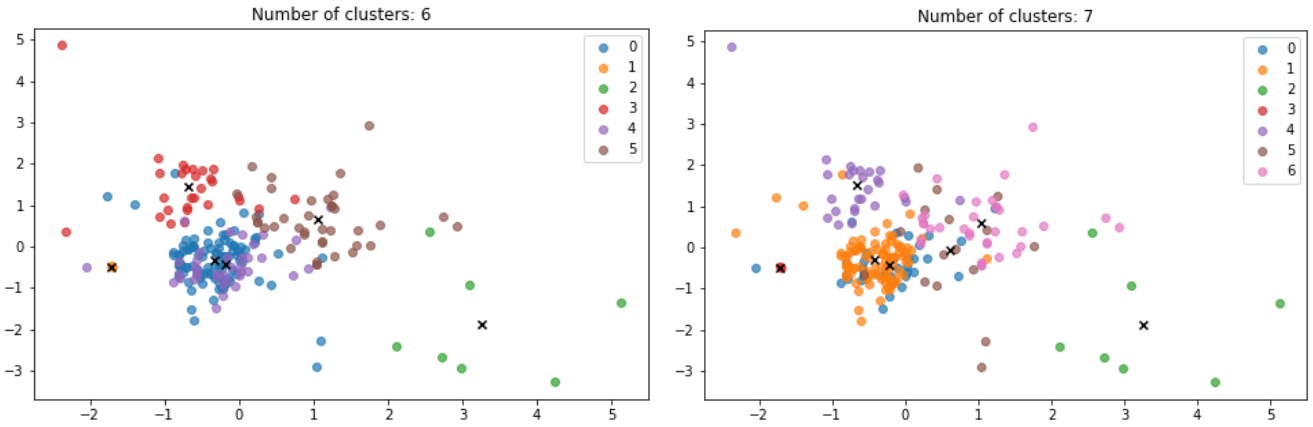
Correlation Heatmap

From the above heatmap, we observed that the features “Refractive Index” and “Calcium” were closely correlated with each other (one of them could be dropped).

Implementing K-Means Algorithm on the dataset

It was performed using `sklearn.cluster.KMeans` class. The number of clusters (the value of k) was varied, and the clusters and their centroids were visualized. We varied the value of k from 2 to 7.



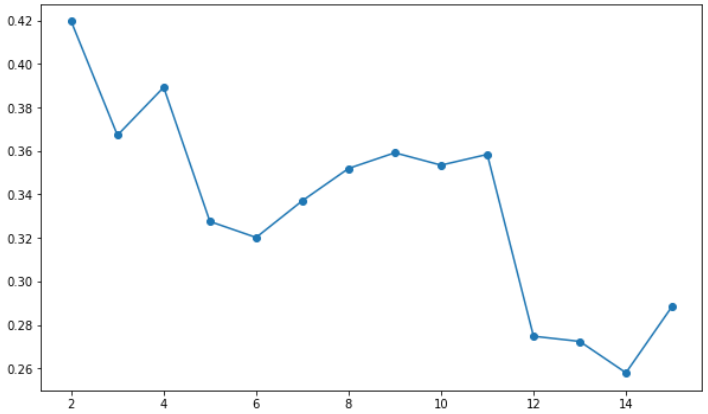


Visualization of the clusters along with their centroids for different values of k

Using "Silhouette score" to find the optimal value of k for clustering

The silhouette score is a metric to analyze how well the clustering is performed. It assigns positive values less than or equal to unity to clusters that are well separated from each other and negative values to poorly separated clusters. Hence, the higher the silhouette score, the better the clustering has been performed.

We found the silhouette scores for k values between 2 and 15. The highest silhouette score was obtained in the case of n_clusters=2 (or k=2). Hence, using this method, the optimal value of k was found to be "2".

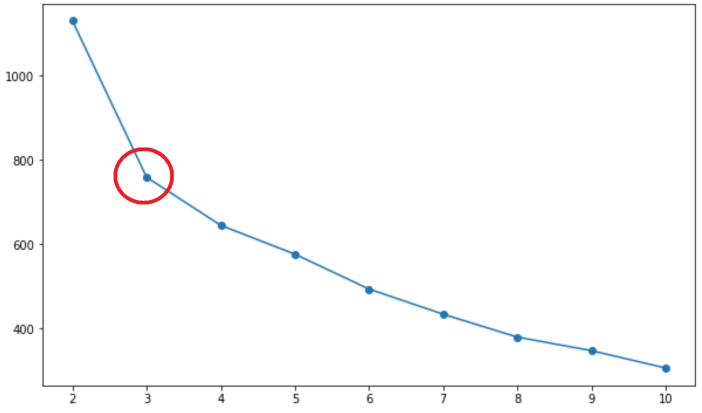


Plot showing the variation of the silhouette score with respect to k (or n_clusters)

Using "Elbow method" to find the optimal value of k for clustering

The elbow method is a technique used to determine the optimal number of clusters in a dataset. In this method, we plot the inertia/distortion of the model as a function of the number of clusters (k). Then, we pick up the elbow of the curve, and that corresponding value of k is considered to be optimal.

The following graph was plotted in the case of this dataset:-



Variation of the inertia of the model with respect to k (or n_clusters)

In the above graph, we observe that k=3 is the elbow of the curve, and after this point, the distortion starts decreasing in a linear portion. Hence, the elbow method gives the optimal value of k as “3”.

Bagging with the KNN classifier as the base model

The `sklearn.ensemble.BaggingClassifier` and the `sklearn.neighbors.KNeighborsClassifier` was used to perform this operation. We defined functions to return the training and testing accuracy scores when the dataset is trained with a KNN Classifier and when it is trained with a Bagging Classifier with KNN as the base model.

The following scores were obtained:-

KNN Accuracies (without Bagging)

```
AccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=1)
```

```
training accuracy: 1.0
testing accuracy: 0.7037037037037037
```

```
AccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=2)
```

```
training accuracy: 0.83125
testing accuracy: 0.6296296296296297
```

```
AccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=3)
```

```
training accuracy: 0.8375
testing accuracy: 0.6296296296296297
```

Bagging Accuracies

```
BaggingAccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=1)
```

```
training accuracy: 0.98125
testing accuracy: 0.7222222222222222
```

```
BaggingAccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=2)
```

```
training accuracy: 0.925
testing accuracy: 0.7037037037037037
```

```
BaggingAccuracyKNN(train_X, train_y, test_X, test_y, n_neighbors=3)
```

```
training accuracy: 0.8875
testing accuracy: 0.6666666666666666
```

In terms of variance and bias, from the above scores, we were able to observe that the KNN model is prone to overfitting (due to the “curse of dimensionality”). The training accuracy was very high compared to the testing accuracy.

However, using the Bagging Classifier with KNN as the base model reduced the variance. The testing accuracy increased considerably. Hence, we could say that the Bagging Classifier did not overfit the data and improved the accuracy. The bias also decreased a bit in the case of k=2 and k=3 since the training accuracy increased as compared to that of KNN.

Question-2

K-Means Clustering Algorithm implementation from scratch

For implementing the K-Means clustering algorithm, we created a class named `KMeansAlgo`. The constructor method took as input the value of k, i.e., the number of clusters and the number of maximum iterations. Also, the constructor was implemented to be able to take initial cluster center points from the user as its initialization.

The following methods were implemented in the class:-

- `fit`
  It took in a training set as input and trained the clustering model by manipulating and storing the cluster centers based on the number of iterations and comparison of euclidean distances of the datapoints.
- `euclidean_distance`
  It took in the centroid and the datapoint and returned the euclidean distance between the two.
- `compare_distances`
  It took a datapoint as input, compared its euclidean distance from each centroid, and returned the cluster number of the centroid with the least euclidean distance.
- `sse`
  It returned the sum of squared error at any instance.
- `predict`
  It took in a test datapoint and returned the cluster to which the datapoint belongs using the compare_distances helper method.
- `convergence`
  It checked the condition of convergence by checking if the current iteration is greater than the maximum iteration provided by the user. A part of this was implemented in the fit method, which stopped the iterations when the centroids stopped changing in consecutive iterations.

Model-1

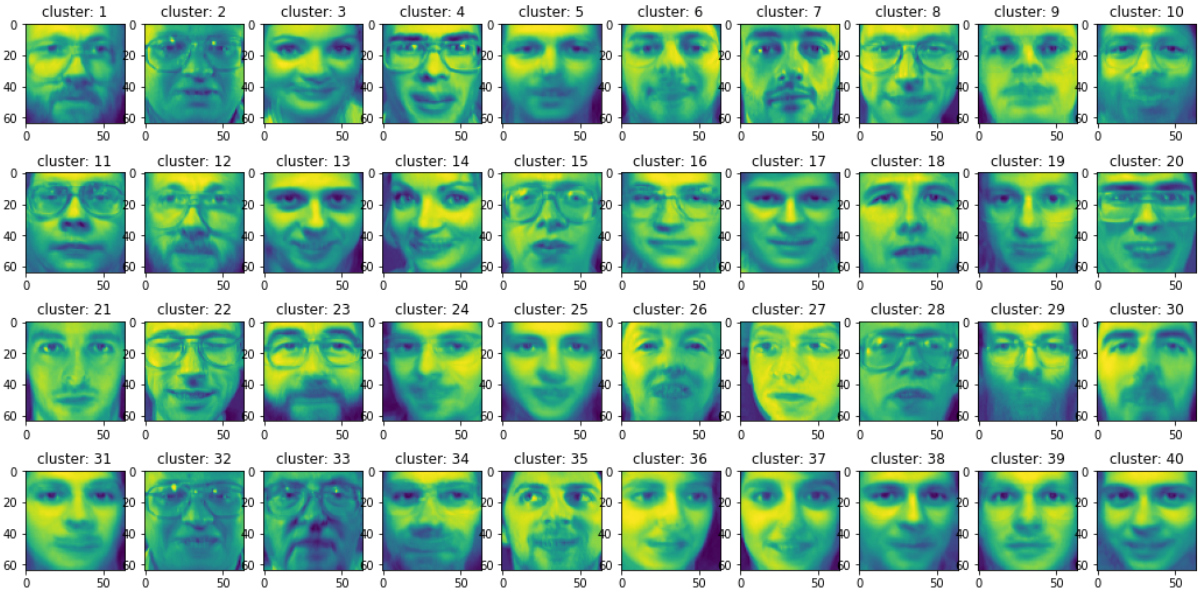
Training the KMeans Algo on the Olivetti dataset *(with k=40 and 40 random 4096-dimensional points as initializations)*

By using the Olivetti Faces dataset from sklearn, we took 40 random data points from the dataset and implemented the KMeansAlgo model on the dataset, taking those 40 points as the initial centroids for the clusters.

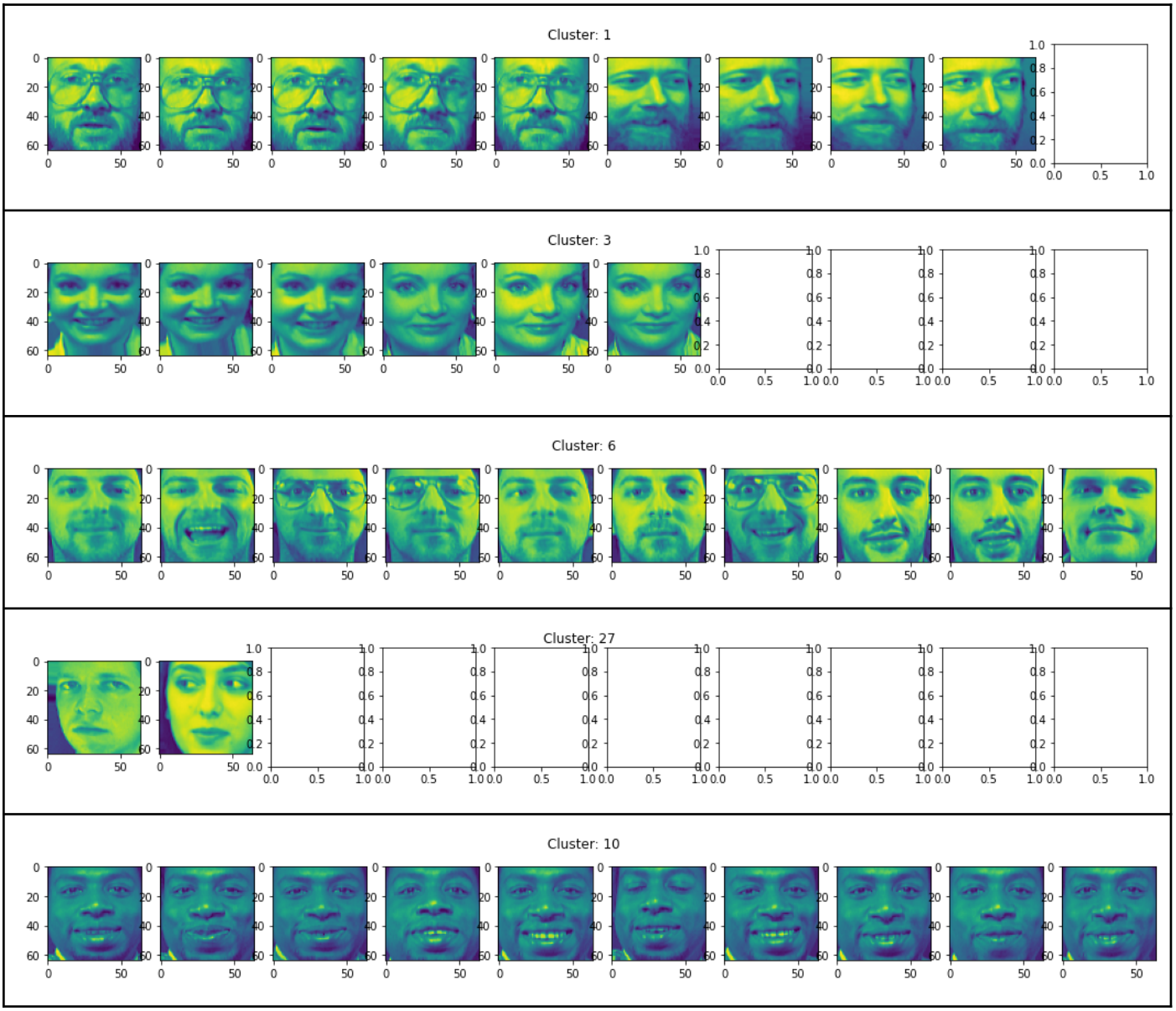
After the completion of the training of the model, the number of the points in each cluster were reported as follows:-

```
 'Cluster-1': 9, 'Cluster-2': 2, 'Cluster-3': 6, 'Cluster-4': 4,
'Cluster-5': 29, 'Cluster-6': 11, 'Cluster-7': 2, 'Cluster-8': 8,
'Cluster-9': 13, 'Cluster-10': 30, 'Cluster-11': 4, 'Cluster-12': 6,
'Cluster-13': 7, 'Cluster-14': 3, 'Cluster-15': 6, 'Cluster-16': 7,
'Cluster-17': 22, 'Cluster-18': 10, 'Cluster-19': 14, 'Cluster-20': 4,
'Cluster-21': 5, 'Cluster-22': 2, 'Cluster-23': 9, 'Cluster-24': 12,
'Cluster-25': 21, 'Cluster-26': 9, 'Cluster-27': 2, 'Cluster-28': 4,
'Cluster-29': 10, 'Cluster-30': 15, 'Cluster-31': 19, 'Cluster-32': 3,
'Cluster-33': 10, 'Cluster-34': 10, 'Cluster-35': 3, 'Cluster-36': 6,
'Cluster-37': 5, 'Cluster-38': 25, 'Cluster-39': 17, 'Cluster-40': 16
```

Visualization of the cluster centers of each cluster as 2D images of all the clusters:-



Visualization of 10 images corresponding to each cluster:-
(5 random clusters are taken here in order to show the performance of the clustering algorithm)



We observe that not all the faces are clustered correctly, but some of them are perfectly clustered. There are also some clusters that show comparatively poorer results.

Model-2

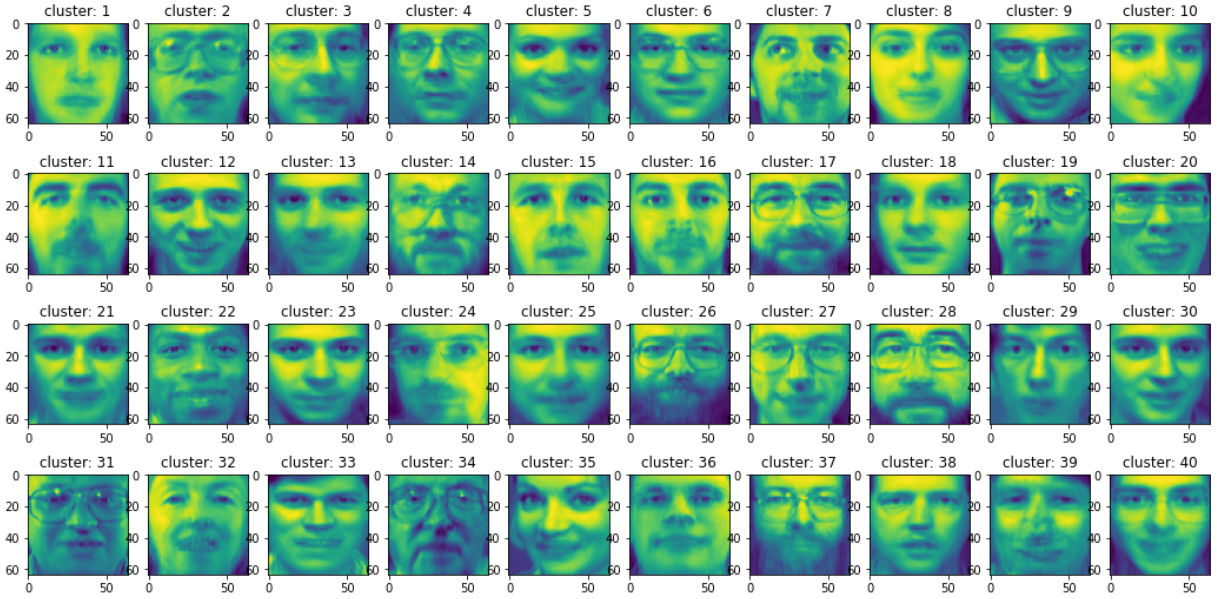
Training the KMeans Algo on the Olivetti dataset *(with k=40 and 40 images, one from each class, as initializations)*

We took one point randomly from each class of the Olivetti Faces dataset and initialized a `KMeansAlgo` model, taking each point as one of the initial centroids.

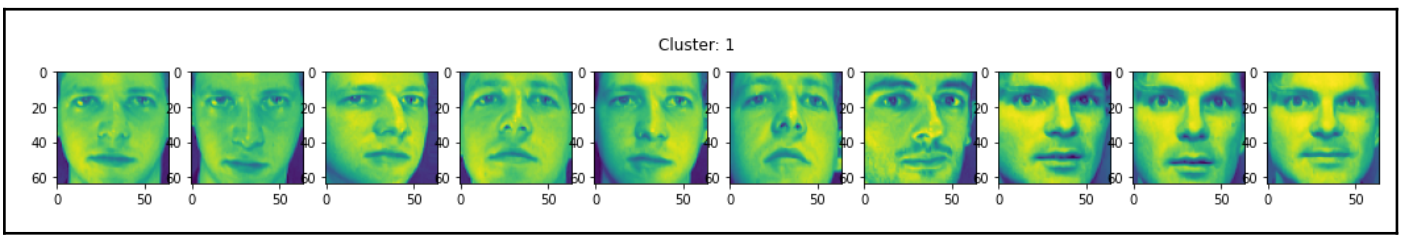
After the completion of the training of the model, the number of the points in each cluster were reported as follows:-

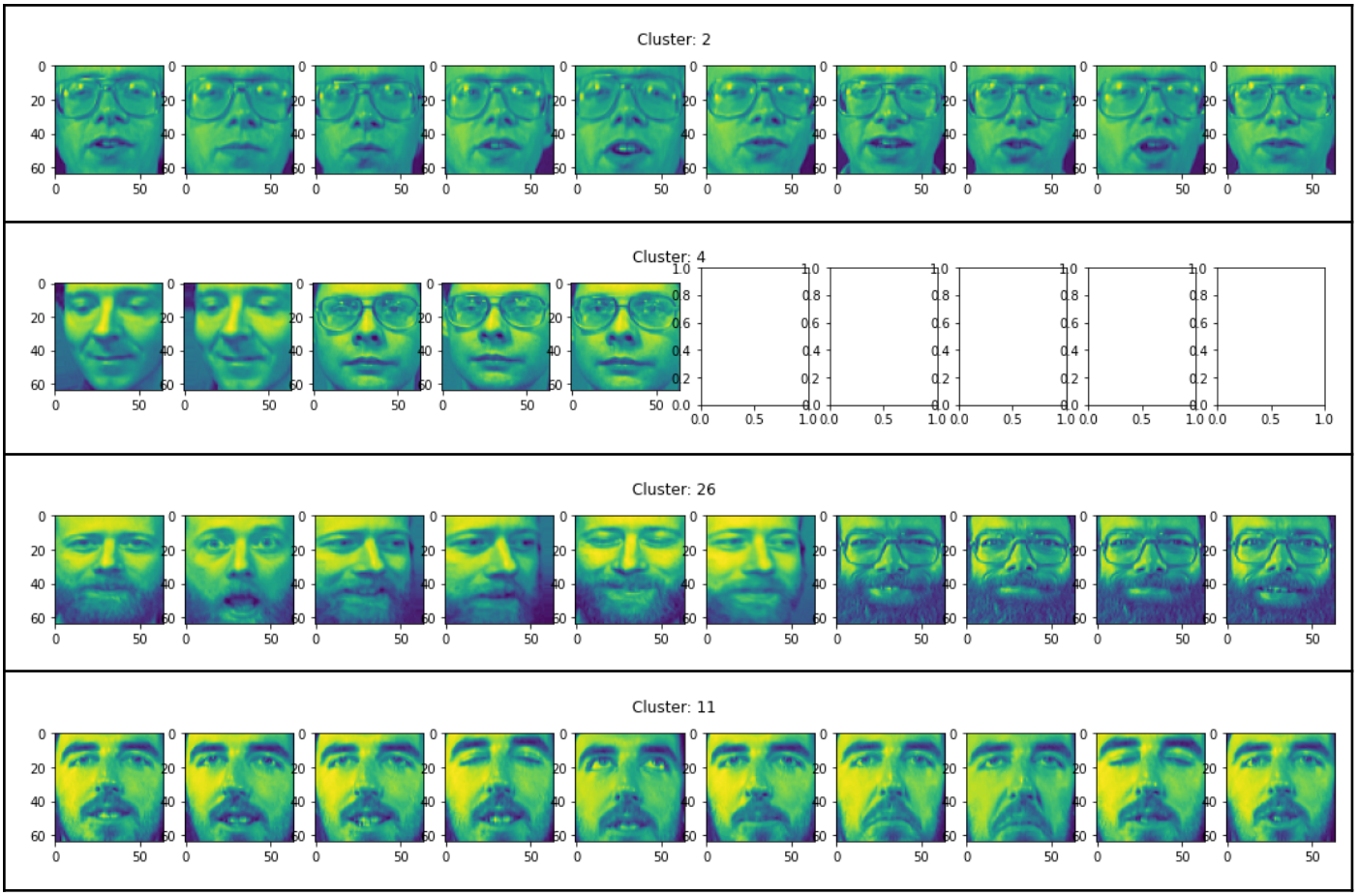
```
'Cluster-1': 10, 'Cluster-2': 10, 'Cluster-3': 8, 'Cluster-4': 5,
'Cluster-5': 10, 'Cluster-6': 10, 'Cluster-7': 3, 'Cluster-8': 7,
'Cluster-9': 7, 'Cluster-10': 8, 'Cluster-11': 15, 'Cluster-12': 5,
'Cluster-13': 23, 'Cluster-14': 10, 'Cluster-15': 17, 'Cluster-16': 11,
'Cluster-17': 6, 'Cluster-18': 12, 'Cluster-19': 5, 'Cluster-20': 4,
'Cluster-21': 16, 'Cluster-22': 13, 'Cluster-23': 11, 'Cluster-24': 9,
'Cluster-25': 28, 'Cluster-26': 11, 'Cluster-27': 10, 'Cluster-28': 5,
'Cluster-29': 10, 'Cluster-30': 11, 'Cluster-31': 5, 'Cluster-32': 10,
'Cluster-33': 10, 'Cluster-34': 10, 'Cluster-35': 4, 'Cluster-36': 11,
'Cluster-37': 6, 'Cluster-38': 10, 'Cluster-39': 7, 'Cluster-40': 17
```

Visualization of the cluster centers of each cluster as 2D images of all the clusters:-



Visualization of 10 images corresponding to each cluster:-
(5 random clusters are taken here in order to show the performance of the clustering algorithm)





We observe that, as compared to the previous model, this time we get better results. Many of the images are perfectly clustered. However, there are also a few clusters that were not properly created. But the overall performance is very high as compared to the previous implementation.

Comparison of the above models using the Sum of Squared Error (SSE) method:
Using the inbuilt function `sse()` of our implemented class KMeansAlgo, we calculated the SSE error for both the models. The errors were reported as:-

- for model-1: *13038.559373050439*
- for model-2: *12481.960467895084*

The sum of squared error is the summation of the sums of the squares of the euclidean distances of all the cluster points from the centroid of the cluster for all the clusters. Hence, the higher this value, the poorer the performance of the model.
From the above errors, it is clear that model-2 performed better than model-1. Hence, the better clustering is in the case of the second implementation of the algorithm.
This is because, in the case of the earlier implementation, we randomly took the points as the centroids for the clusters. However, in the later implementation, we took a point from each class as one of the centroids. Hence, there was a higher probability of that point being nearer to the actual centroid for the cluster representing that particular class, and therefore, the performance was better.

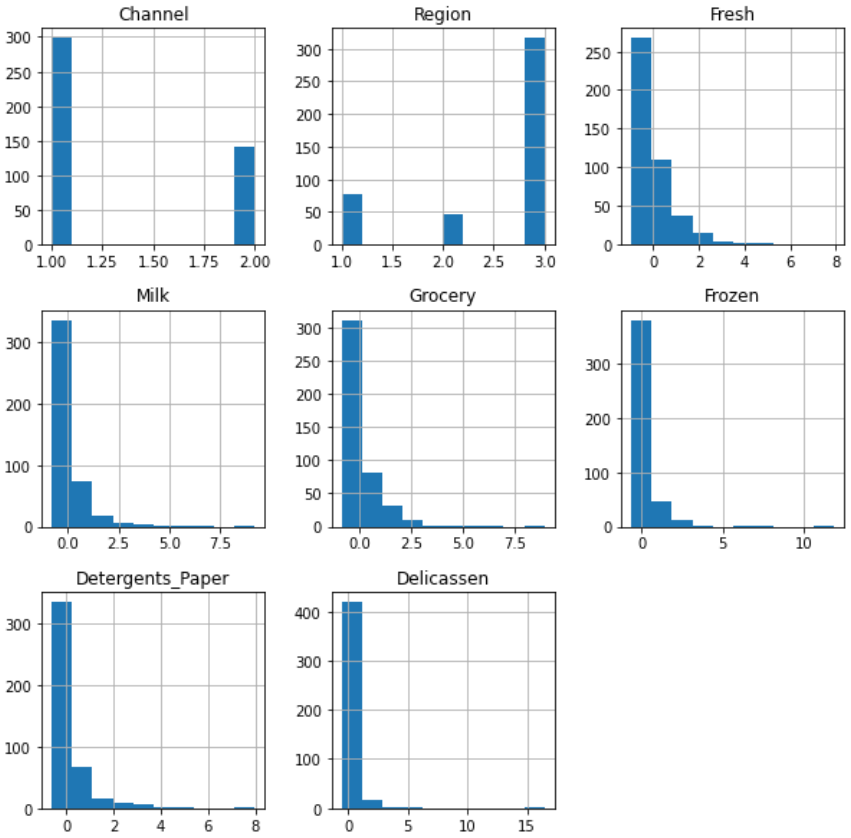
Question-3

Analysis, Preprocessing, and Scaling of the Dataset

The dataset did not contain any NULL entries or NaN values. All the features had differences in their ranges, and hence, they were normalized.

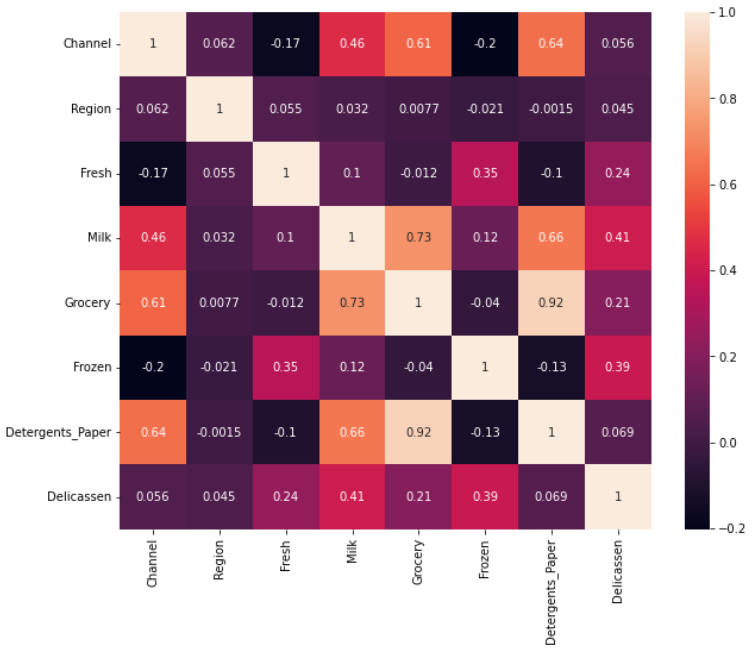
```
Number of unique entries in Channel : 2
Number of unique entries in Region : 3
Number of unique entries in Fresh : 433
Number of unique entries in Milk : 421
Number of unique entries in Grocery : 430
Number of unique entries in Frozen : 426
Number of unique entries in Detergents_Paper : 417
Number of unique entries in Delicassen : 403
```

The number of unique entries corresponding to each feature/label



Histogram plots for the dataset

After normalization, the covariance heatmap for the dataset was plotted. (It was almost identical to the correlation heatmap.)

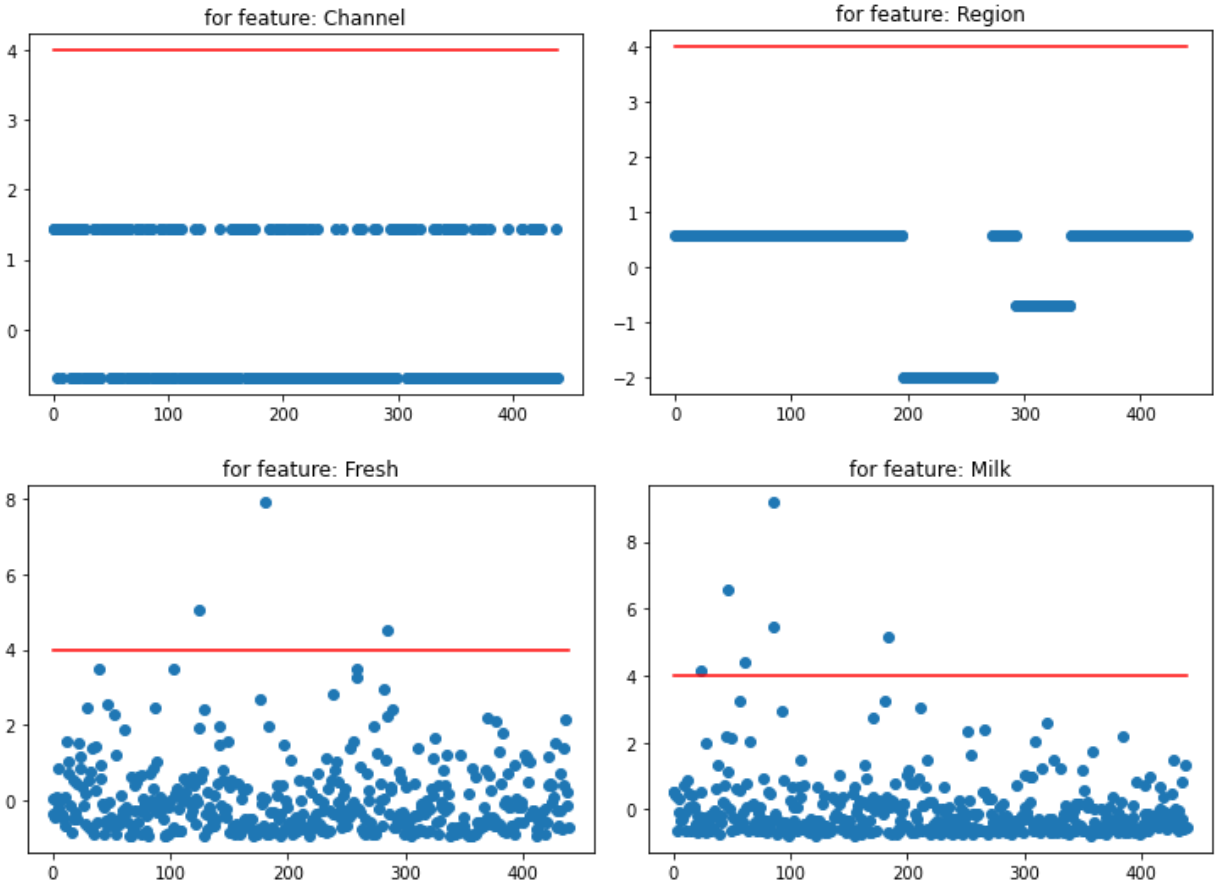


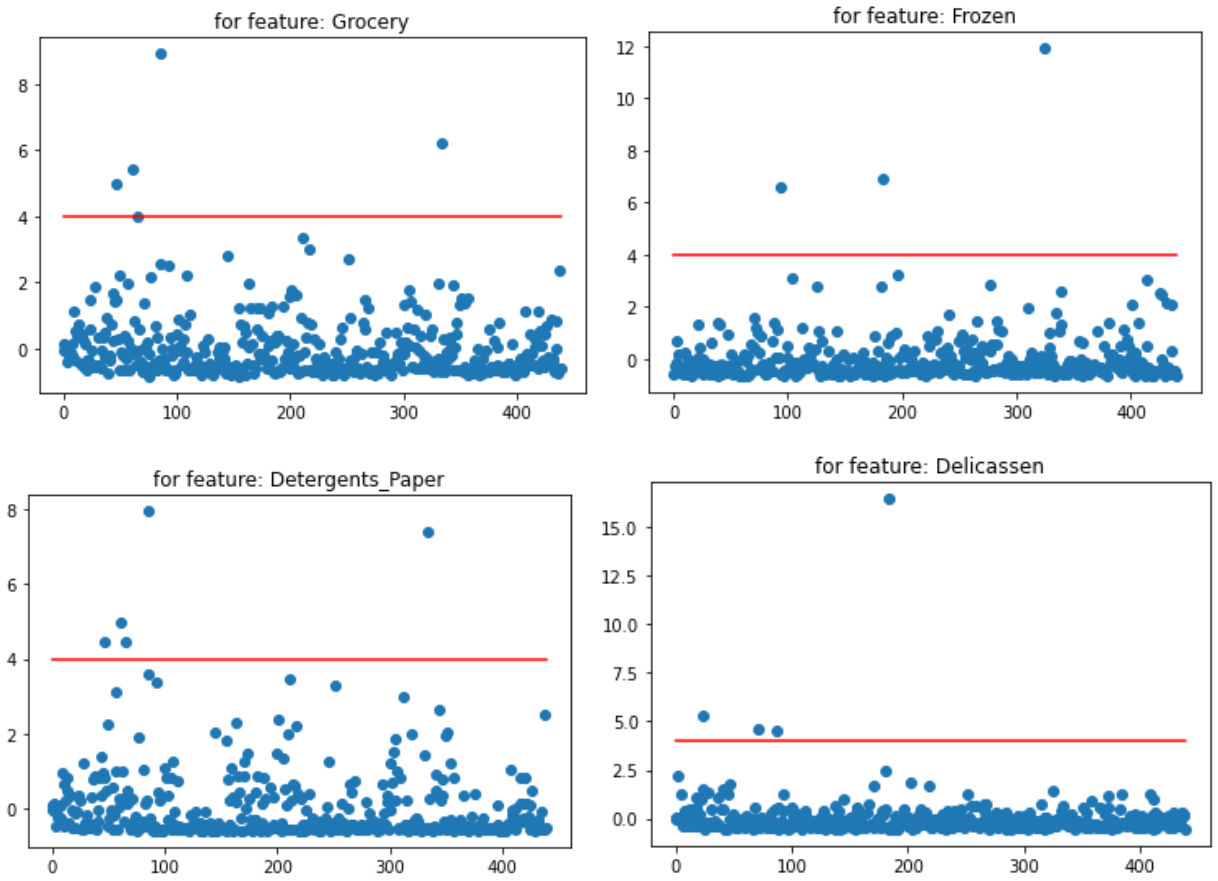
Covariance heatmap of the dataset (after normalization)

Hare, we noted that covariance is highest between the features “Detergents_Paper” and “Grocery”. We considered these two features for outlier visualizations.

Outlier Detection

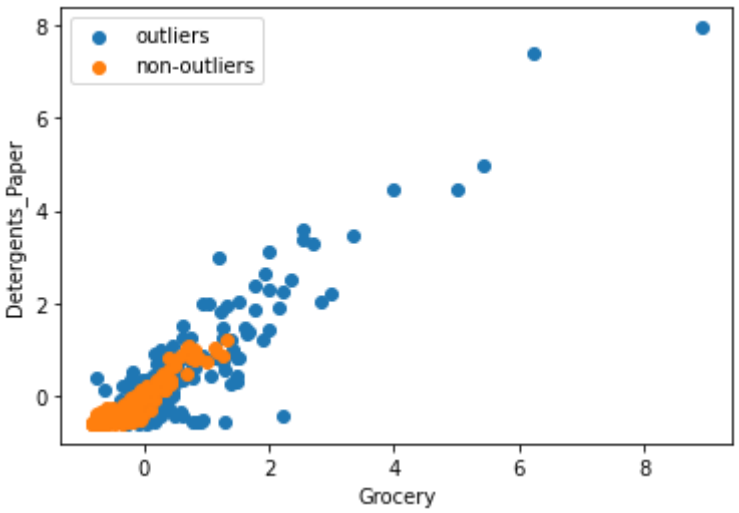
For outlier detection, we initially calculated the Mahalanobis distance for each data point in the dataset. Then, we defined a threshold variable and assigned it some value. Then, we plotted the scatterplots of all the data points along with the threshold line. The following plots were obtained for each feature. The points above the threshold lines were considered outliers.





Outlier detection corresponding to the dataset features

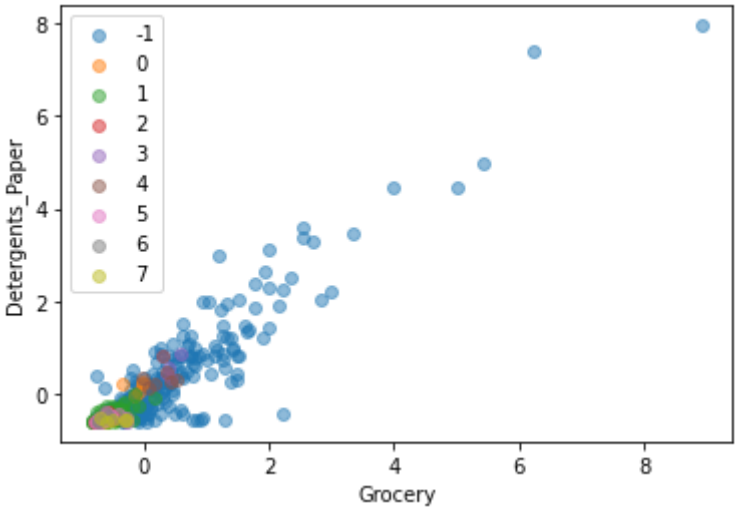
Outliers Visualization



DBSCAN to cluster the dataset

The `sklearn.cluster.DBSCAN` class was used to perform the operation.

The following plot for the clusters was obtained when using DBSCAN for clustering the dataset:-

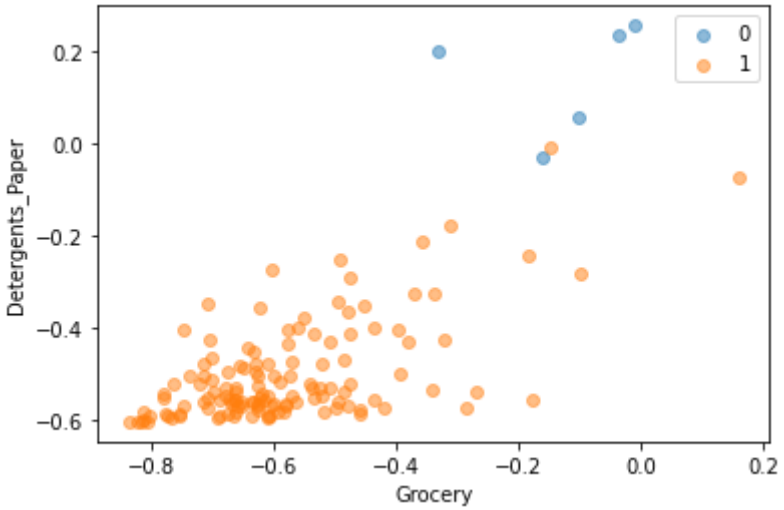


Plot showing DBSCAN Clusters for the dataset

KMeans to cluster the dataset

The `sklearn.cluster.KMeans` class was used to perform the operation.

The following plot for the clusters was obtained when using K-Means algorithm for clustering the dataset:-



Plot showing K-Means Clusters for the dataset

Comparison between DBSCAN and K-Means visualizations

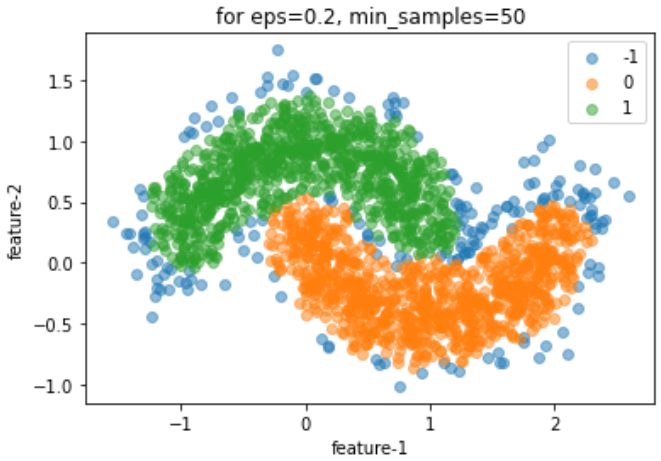
From the graphs, it is evident that that K-Means algorithm is effectively able to distinguish the outliers in this case. However, in case of DBSCAN, the algorithm makes many unnecessary and undistinguishable clusters and hence, it is not a good choice for clustering in this case.

DBSCAN & KNN clustering on the make_moons dataset

Dataset creation was performed using the `sklearn.datasets.make_moons()` function. The dataset consisted of 2000 points, with noise added with a 20% probability.

We made two functions to tune the DBSCAN and K-Means models' hyperparameters and draw the corresponding graphs. By hit-and-trial and looping through the values of the hyperparameters “eps” and “min_samples” for DBSCAN and “n_clusters” for K-Means, we found some of the optimal values for the hyperparameters for DBSCAN. We took the n_clusters value for K-Means to be 2 (since we already know that there are 2 moons in the dataset).

The following plots were obtained after applying DBSCAN and K-Means clustering to the dataset:-



Plot corresponding to DBSCAN clustering

Plot corresponding to KMeans clustering

From the above plots, we observed that the DBSCAN clustering algorithm was proficient in detecting the outliers in the dataset (the random noise points we appended while creating the dataset). However, in the case of the KMeans clustering algorithm, it was observed that the algorithm was unable to detect the outliers. Hence, it did not give optimal results in comparison to the DBSCAN algorithm.
Also, the K-Means algorithm couldn't put the two moons in the dataset into the right groups. This is because the K-Means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).