Pattern Recognition and Machine Learning

Lab - 6 Assignment

Akriti Gupta
B21AI005

Question 01:

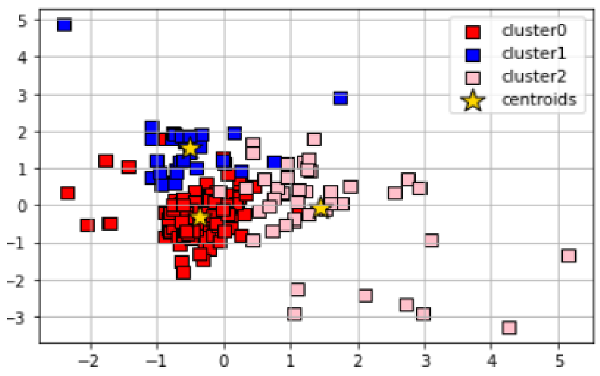
Dataset: Glass Classification Dataset

Preprocessing:

First, we import all the necessary libraries and then import the glass dataset. Now we start the preprocessing of dataset by first checking if there are any null values present in it or not. As no missing values are there we proceed further and check the datatype of each attribute and then also print the description of the dataset. As we are doing unsupervised learning so we don't need the target column so we drop it from the dataset.
Next, we scale the features using StandardScaler() method which helps to ensure that all features are on a similar scale.

a)

For this question, we use the sklearn library to build a kmeans clustering algorithm by taking the value of n_clusters=3 and then find the labels of each datapoint by fitting the glass dataset into themodel and predicting its class using fit_predict() method.
Now, to visualize the clusters, we we first plot all the datapoints by visualizing them according to their classes with the help of different colors i.e cluster 0 is represented by red, cluster 1 by blue and cluster 3 by pink. Next we plot the cluster centres which we found through the class k_means and by using its attribute cluster_centres_ and mark them in the plot using stars. We plot the datapoints by using first two attributes of the dataset.

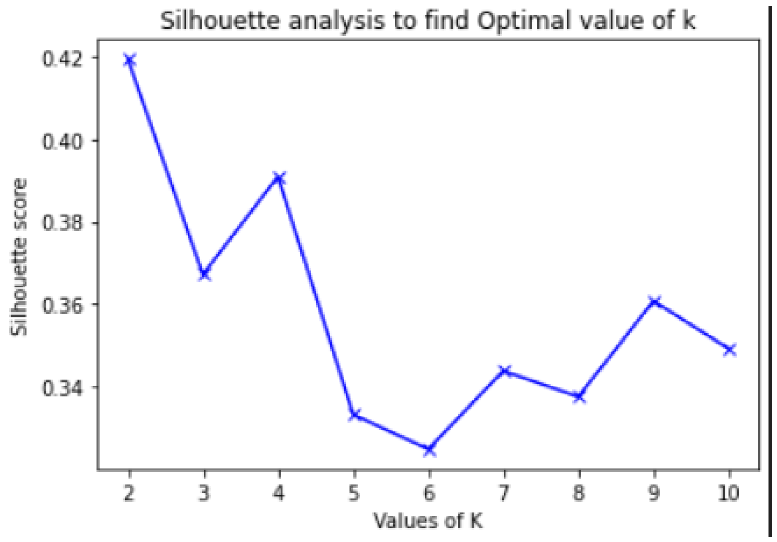


b)

Now we vary the value of k and find the silhouette score for each model. We get the scores as follows:

```
silhouette_score for n_clusters= 2 is 0.41948052343085984
silhouette_score for n_clusters= 3 is 0.3671942443096343
silhouette_score for n_clusters= 4 is 0.3909627794318319
silhouette_score for n_clusters= 5 is 0.3332030725545686
silhouette_score for n_clusters= 6 is 0.3247494403575592
silhouette_score for n_clusters= 7 is 0.34373134578788256
silhouette_score for n_clusters= 8 is 0.3374672401824939
silhouette_score for n_clusters= 9 is 0.3607634576579404
silhouette_score for n_clusters= 10 is 0.34913152071783116
```

The plot for the values of k vs silhouette score in order to find the most optimal value of k is as follows:

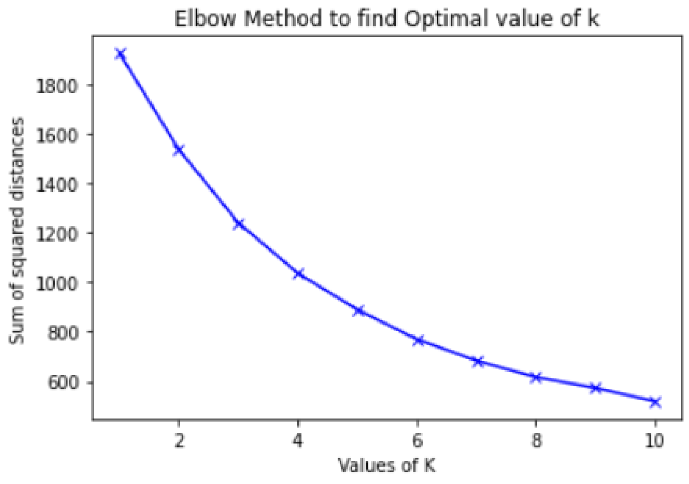


From here, we can observe that the most optimal value for k is 2 as silhouette score is maximimum.

We know that If the silhouette score has a score of 1, it indicates that data points are highly concentrated within their cluster and are situated at a significant distance from other clusters, implying that this is the best scenario.

c)

Now, we implement the elbow method by altering the number of clusters (n_clusters) from 1 to 10 and calculate WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid for n_clusters (using the model's inertia_ method), and then the resulting curve as depicted below to find the best value of n_clusters.



From the above plot we can notice that at k=6 we get the elbow point and hence we conclude that the optimal number of clusters is 6.

d)

Now, we split the dataset into testing and training in the ratio 30:70 to apply knn classification and then bagging algorithm using knn classifier as base model.
We vary k from 1 to 3 and observe the accuracy scores as below:

```
The accuracy score of knn classifier for k= 1 for training dataset= 1.0
The accuracy score of knn classifier for k= 1 for testing dataset=
0.7692307692307693
The accuracy score of knn Bagging for k= 1 for training dataset=
0.9798657718120806
The accuracy score of knn Bagging for k= 1 for testing dataset=
0.7538461538461538


The accuracy score of knn classifier for k= 2 for training dataset=
0.7986577181208053
The accuracy score of knn classifier for k= 2 for testing dataset=
0.7384615384615385
The accuracy score of knn Bagging for k= 2 for training dataset=
0.9328859060402684
The accuracy score of knn Bagging for k= 2 for testing dataset=
0.7230769230769231
```

```
The accuracy score of knn classifier for k= 3 for training dataset=
0.8053691275167785
The accuracy score of knn classifier for k= 3 for testing dataset=
0.7384615384615385
The accuracy score of knn Bagging for k= 3 for training dataset=
0.8389261744966443
The accuracy score of knn Bagging for k= 3 for testing dataset=
0.7384615384615385
```

For k=1:
Upon applying bagging the training score reduced and thus bias becomes less, also the variance(the difference in testing and training score) is reduced so we can say for k=1 upon applying bagging the overfitting reduced.
For k=2:
Upon applying bagging the training score improved a lot, the bias increased but the variance also increased so we can say the model performed better on training data but didn't have better results on the testing data.
For k=3:
Again, we can see that the bias increased but there was not much improvement in testing results, so the variance increases upon applying bagging for k=3.

Question 02:
Dataset: Olivetti Dataset

Preprocessing:
We first import the dataset by using the sklearn.datasets library and divide it into x and y. Next we scale the attributes of dataset by using StandardScalar() method and also observe the count of each class.

a),b)
In order to implement the k-means algorithm from scratch we define a class named k_means. Under it we have a constructor which contains the following attributes:

1. Centres: list which contains the cdentre of each cluster. This is first provided by the user.
2. Datapoints: the dataset on which this model is to be trained. This is also given by the user.
3. K: The number of clusters the model will make. This is also given by the user.
4. Iter: The no. of max iterations before the model converges i. E. before the previous sets of centroids coincide with the next. If max value is achieved the algo stops and returns the centroids and labels of the last iteration.

Next, the class has the following attributes:

1. eucl: this function returns the euclidean distance between two datapoints
2. dist: gives a list containing euclidean distance of a list of points from a center.
3. Lis: returns a list containing k lists where each list contains distance of all datapoints from a specific center.
4. Groups: returns the label of every datapoint.
5. Centroids: returns the centre list and the labels
6. Run: calls the centroids function and updates the self.centres attribute of the class and also returns the centres and labels wrt the dataset.
7. Predict: calls the run function and returns the labels obtained from it.

c)

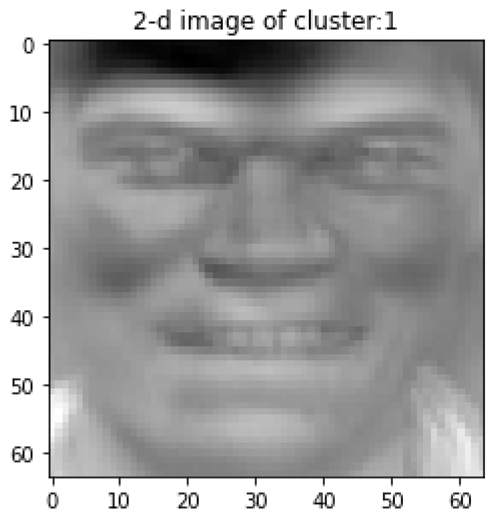
We take 40 random points from the data as cluster centres initially and make an object of our k_means model. Following are the number of points in each clusters:

```
{23: 12,
    30: 6,
    29: 5,
    25: 15,
    12: 19,
    16: 13,
    21: 20,
    19: 12,
    11: 21,
    14: 10,
    20: 23,
    15: 20,
    9: 27,
    31: 8,
    17: 6,
    18: 7,
    5: 22,
    7: 8,
    36: 7,
    39: 18,
    26: 5,
    28: 5,
    10: 7,
    32: 5,
    33: 17,
    38: 4,
    34: 7,
    4: 9,
    24: 4,
    22: 4,
    35: 5,
    8: 5,
    1: 9,
    27: 10,
    0: 3,
    6: 2,
    13: 8,
    3: 5,
    37: 6,
    2: 1}
```

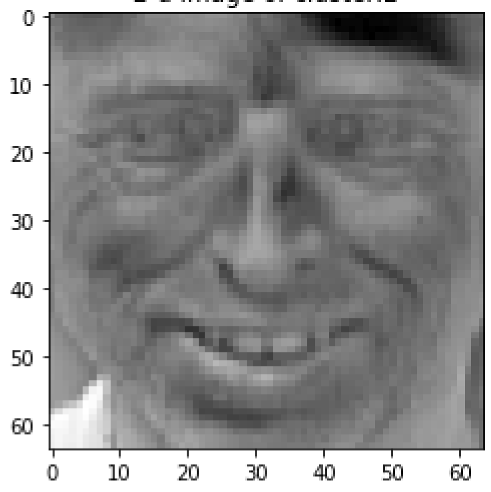
Key_value pair=(Cluster_label: number of points in that cluster)

d)

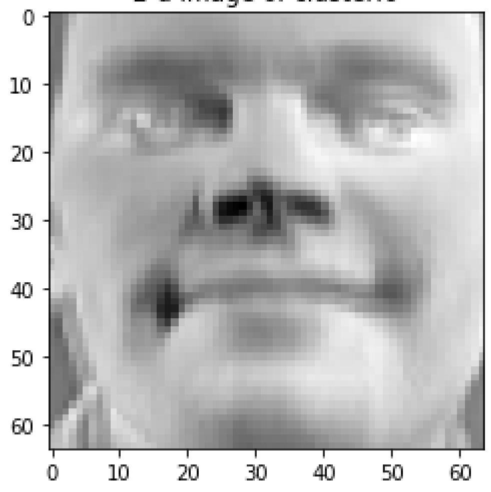
For this part we reshape each of our cluster centres into a 2D array of dimension 64x64. Now, we use imshow from matplotlib to visualize the cluster centre images as shown below:



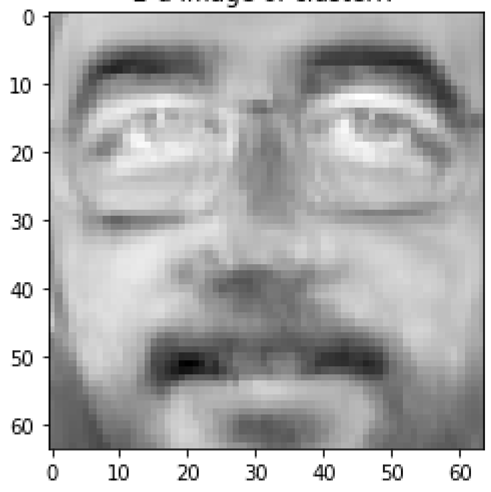
2-d image of cluster:2

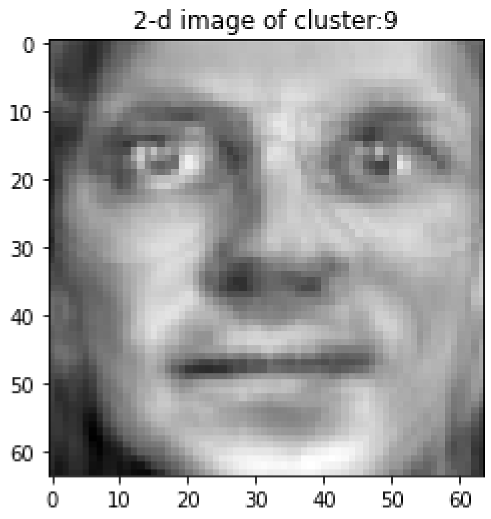


2-d image of cluster:6



2-d image of cluster:7





These were few of the cluster centres, there are 40(n_clusters) such images in the colab file.

e)

For visualizing 10 images from each clusters we group the datapoints with same cluster label together and plot atmost 10 images per cluster as shown:

For cluster 1:

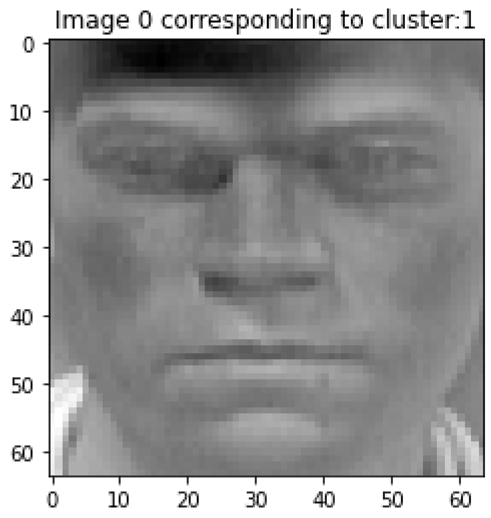


Image 1 corresponding to cluster:1

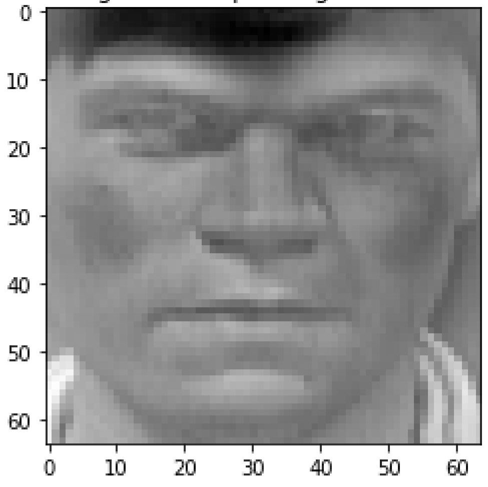
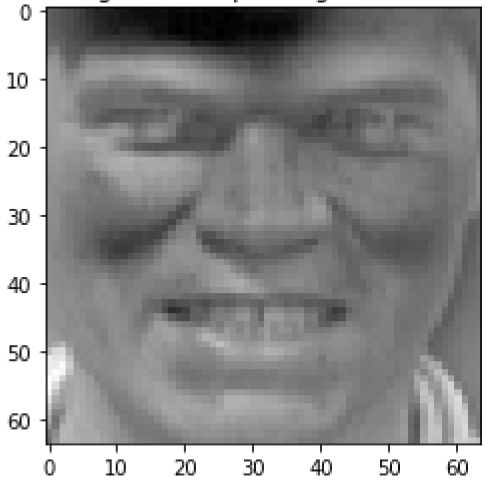
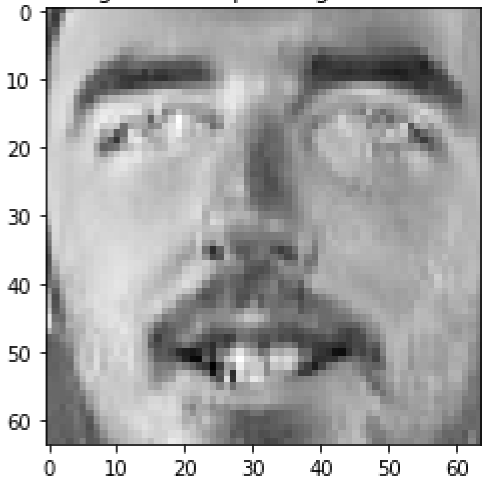


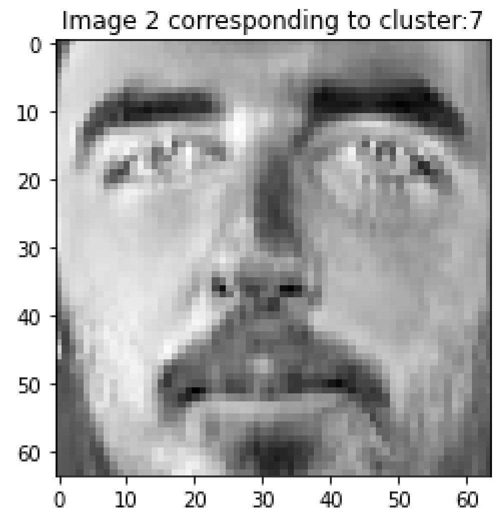
Image 2 corresponding to cluster:1

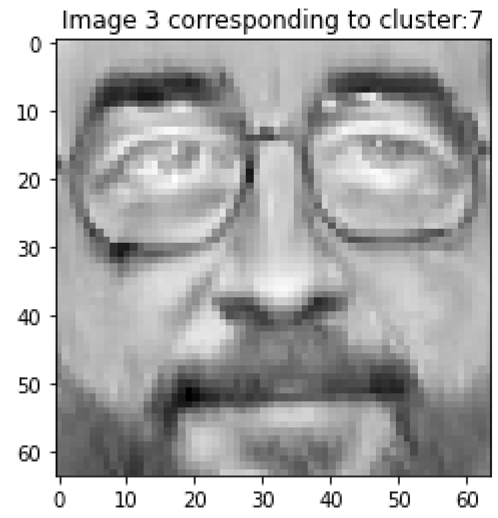


For cluster 7:

Image 1 corresponding to cluster:7







Similarly one can view more such images corresponding to each clusters from the colab file.
Observing the images, we can see that each of the images in a cluster is of the same person as that in the corresponding cluster centre, and hence we can conclude that the clustering was very efficient.

f)

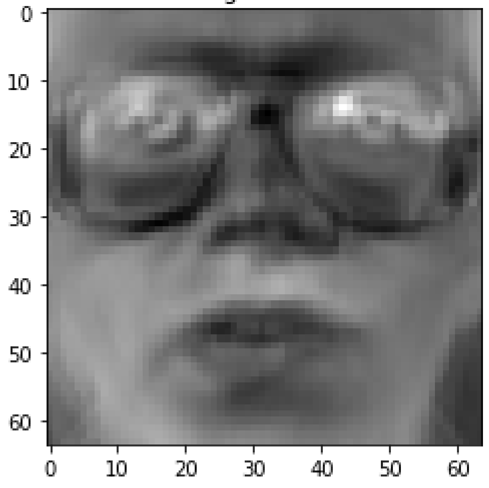
For this part, we take 1 image from each label(last image with that target value in the dataset) and thus have 40 such points(as there are 40 labels) as cluster_centres initially, and then train our k_means model. Following are the number of points in each clusters:

```
{0: 11,
   7: 6,
   37: 24,
   17: 20,
   11: 8,
   1: 10,
   2: 14,
   30: 17,
   4: 19,
   3: 5,
   22: 16,
   14: 6,
   24: 23,
   5: 10,
   6: 7,
   8: 11,
   39: 9,
   9: 3,
   10: 15,
   12: 11,
   13: 10,
   15: 2,
   16: 6,
   18: 5,
   19: 4,
   29: 15,
   21: 10,
   23: 13,
   26: 10,
   27: 10,
   28: 6,
   20: 5,
   31: 10,
   32: 10,
   33: 10,
   34: 3,
   35: 6,
   36: 5,
   25: 5,
   38: 10}
```

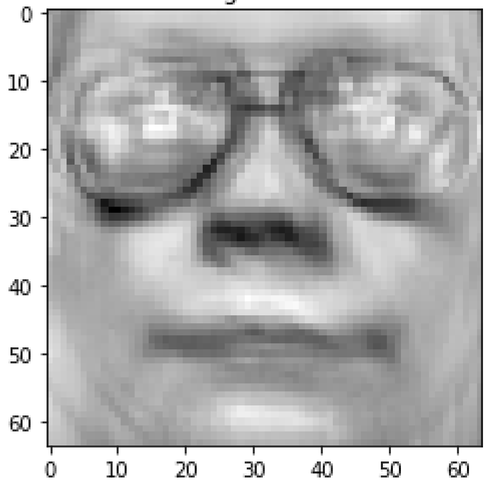
Key_value pair=(Cluster_label: number of points in that cluster)

Visualizing some of the cluster_centres:

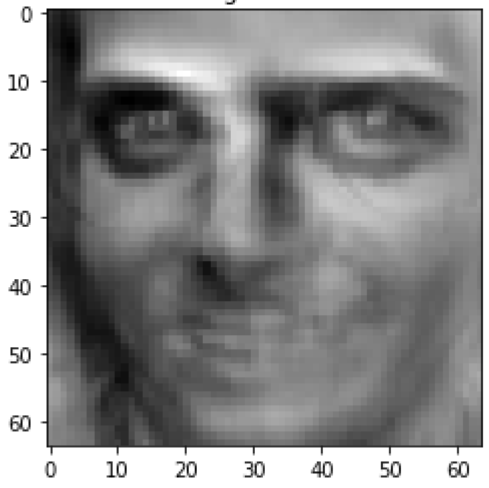
2-d image of cluster:1

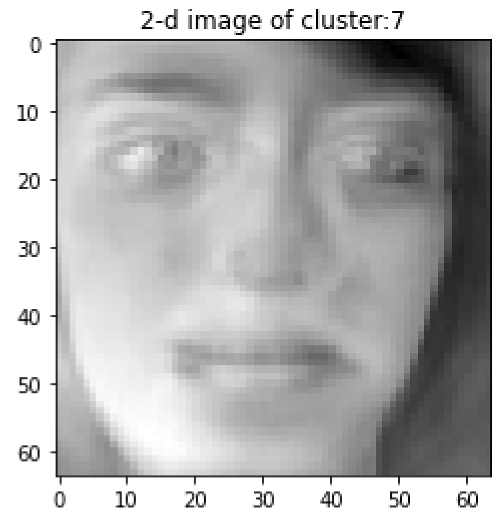


2-d image of cluster:3



2-d image of cluster:4

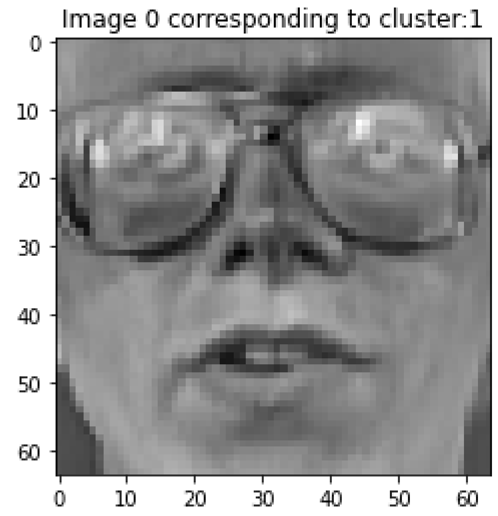


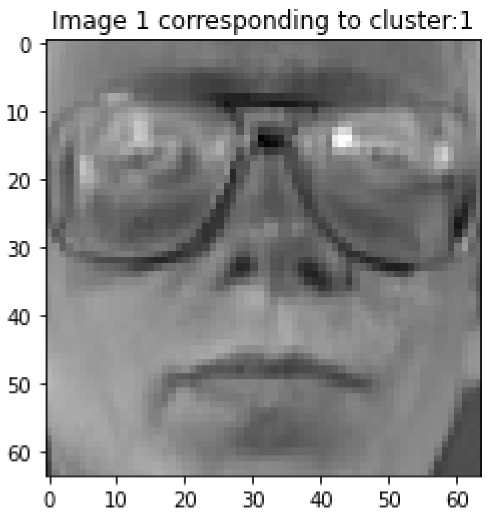


We can find 40 such cluster centres in the colab file.

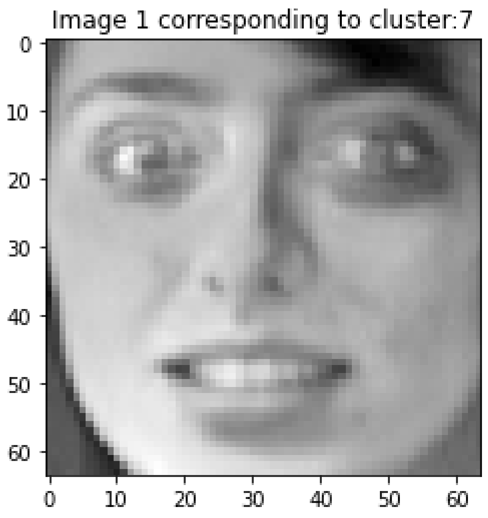
g)

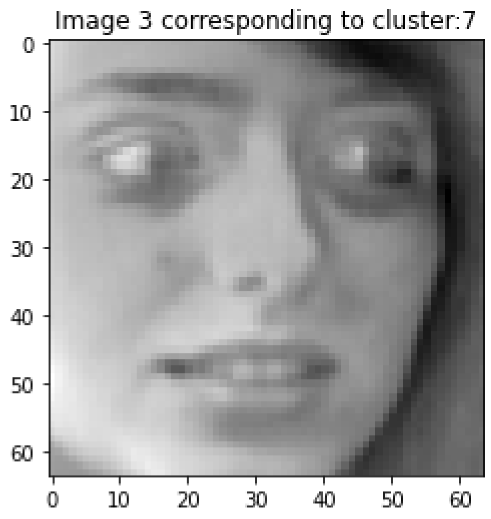
We visualize atmost 10 points corresponding to each clusters obtained in part f.
For cluster 1:





For cluster 7:





Similarly one can view more such images corresponding to each clusters from the colab file.
Observing the images, we can see that each of the images in a cluster is of the same person as that in the corresponding cluster centre, and hence we can conclude that the clustering was very efficient.

h)

Now, for sse we use sse function as defined in the colab file and get the following results:
For part c:
sse=645164.062986185

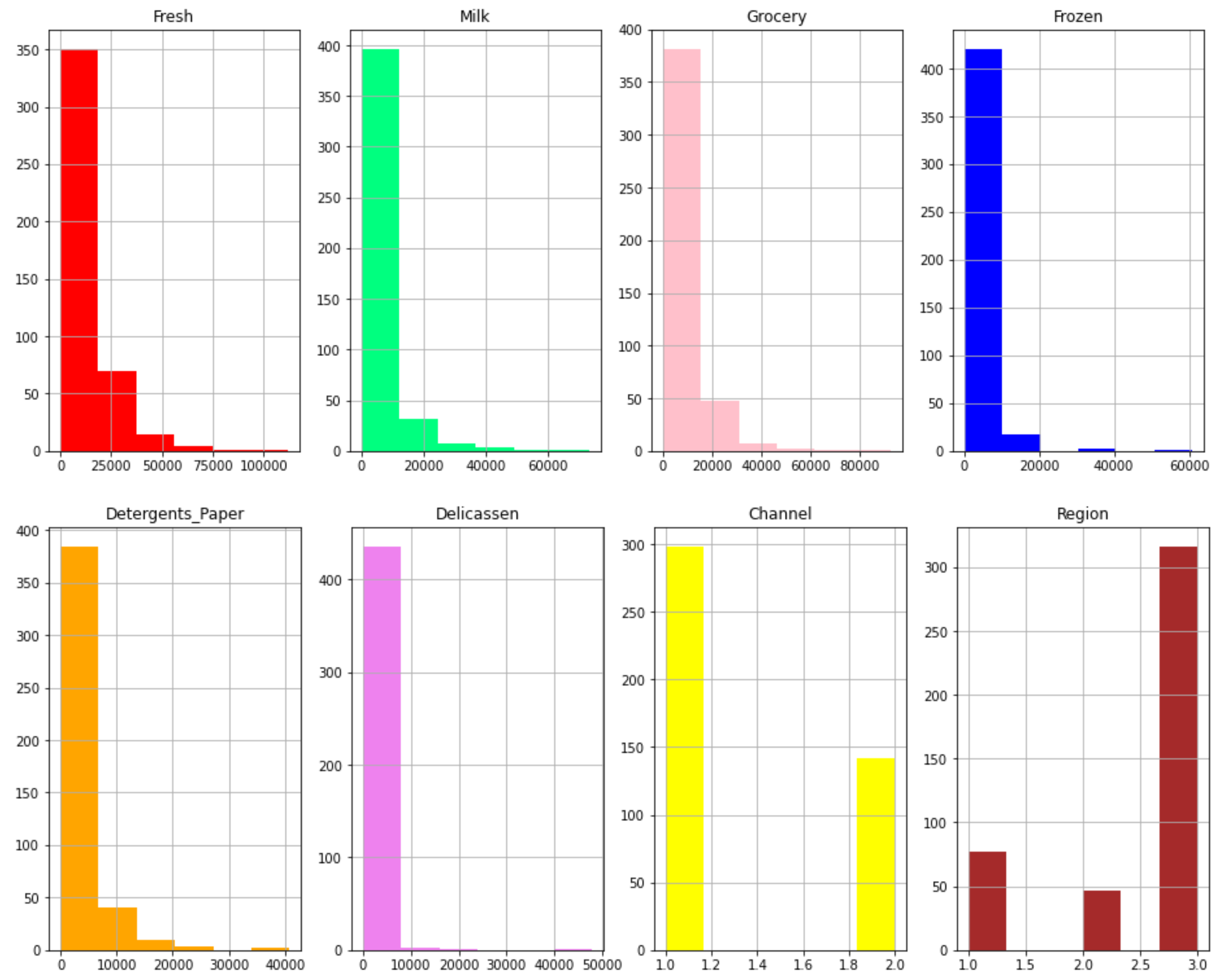
For part f:
sse=618831.4235038817
So, observing the sum of squared errors, part f is better as it gives lower sse. So the case of part f is a better clustering.

Question 03:
Dataset: Wholesale Customers Dataset and make_moons dataset

a)

<u>Preprocessing</u>

We use StandardScaler to scale the features to same scale. For visualizing the data we plot histograms wrt each features:

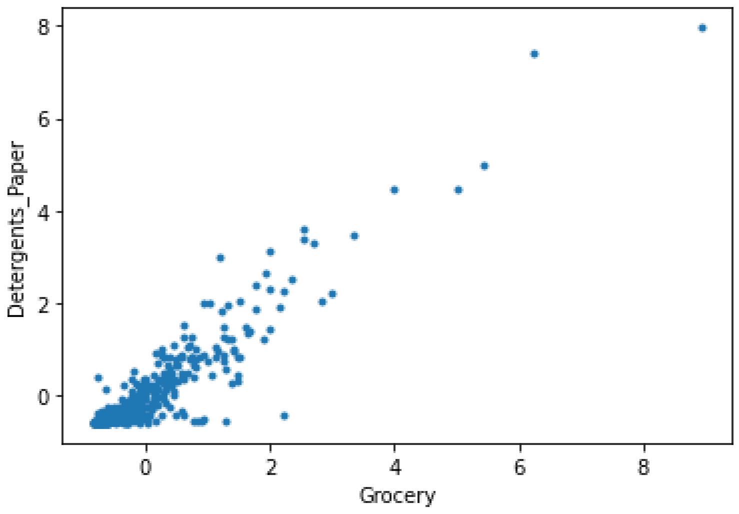


b)

To find the best pair of features to visualize the outliers we find the features which are linearly related to each other. For that we check the covariance of each pair of feature and obtain the pair with maximum covariance using best_features functions in the code.

```
Features with highest covariance Grocery and Detergents_Paper and they
have covariance= 0.9267469338858263
```

Upon visualizing the pair of features we get the following scatter plot:



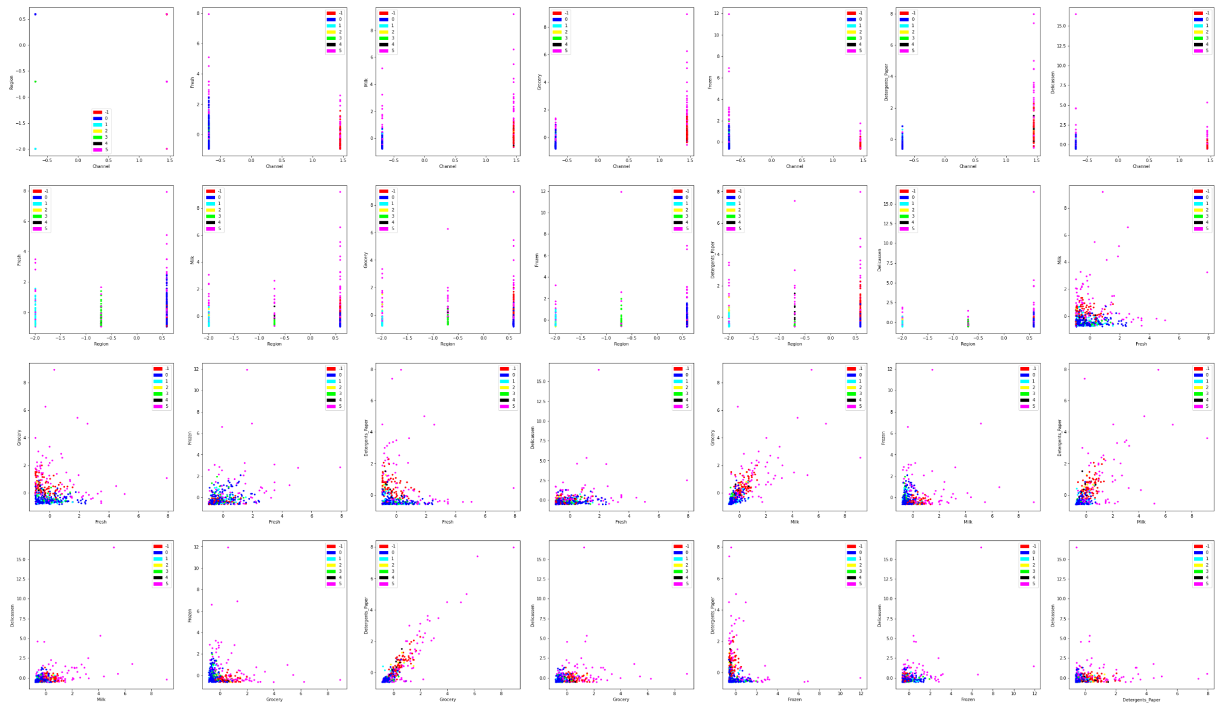
So, we can clearly see that the features are nearly linearly related to each other and hence we can visualize the outliers the best.

c)

We apply DBSCAN from sklearn with eps=1 and min_samples=5 and obtain the cluster labels corresponding to each point using labels_ attribute.

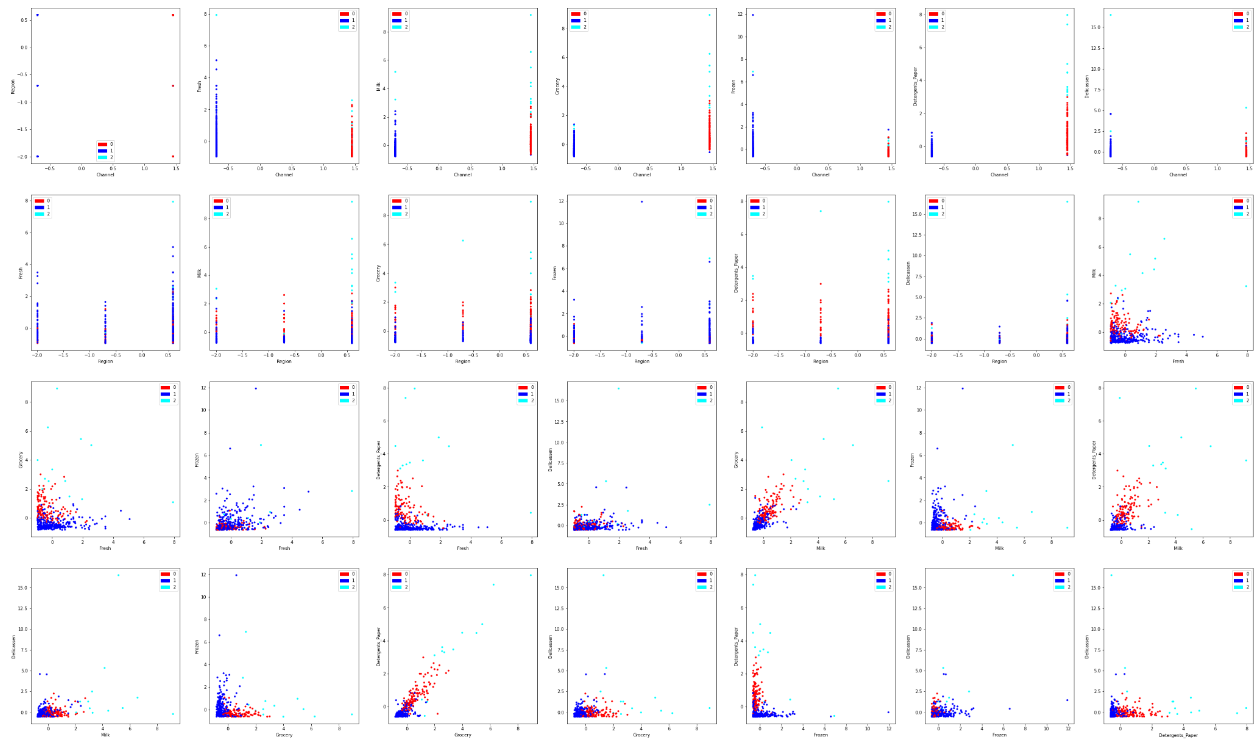
Now, we visualize the clusters and colour each points according to their labels using the colormap defined in the code.

We make scatter plots wrt every pair of features as shown:



d)

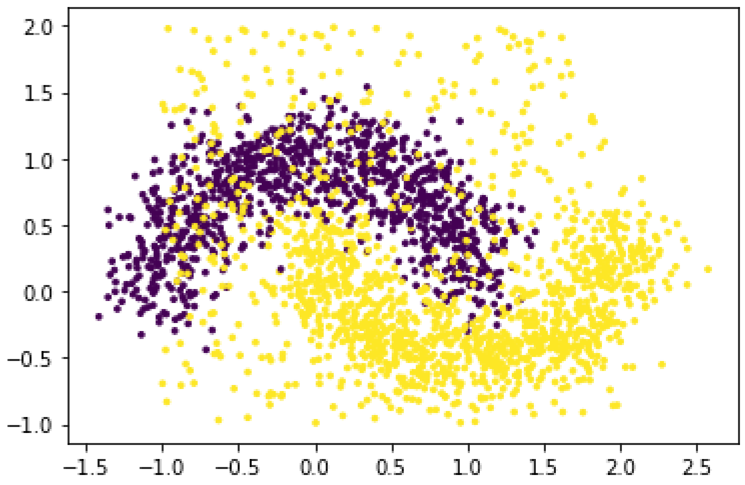
We apply KMeans from sklearn on the same dataset with n_clusters=3 and use the same technique to make scatter plots as in part c.
The visualization plots are:



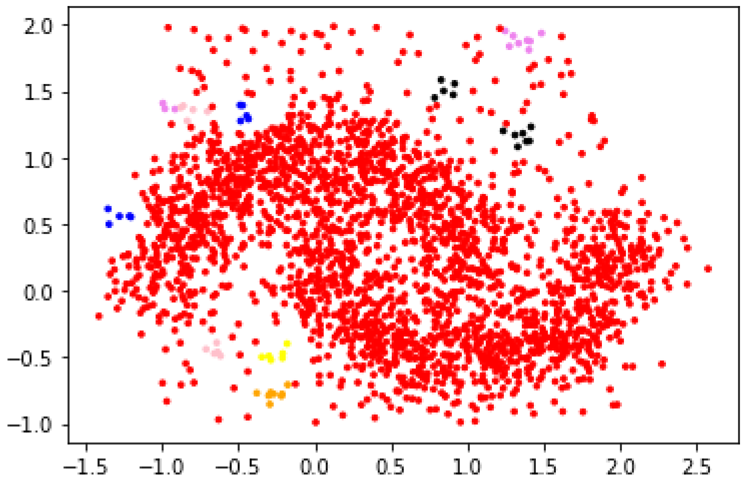
So, we can observe clearly that DBSCAN is able to identify the outliers in the data and colour them differently showing that they don't belong to any of the clusters, as we can clearly see in plot of Detergent_Papers vs Grocery, the outliers are coloured differently. Whereas in case of KMeans the outliers are also grouped in a cluster along with the other core points as visible from Detergent_Papers vs Grocery plot which shows that KMeans is not a good algorithm to deal with outliers.

e)

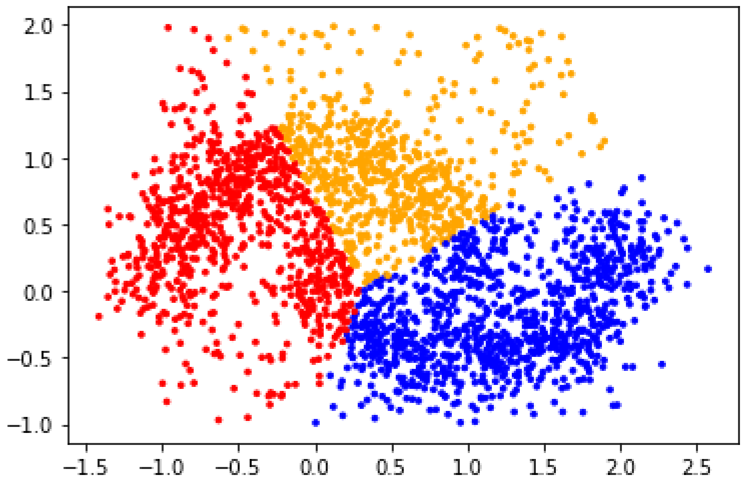
For this part we use make_moons library and create a dataset with n_samples=2000 and noise=0.2.
Now, we add random data points with probability of 0.2 and get the following data set:



Now, we use DBSAN with eps=0.1 and min_samples=5, fit the model on data set and get the cluster labels of each points.
Upon visualizing the clusters we get the following scatter plot:



Now, we apply KMeans with n_clusters=3 and get the following scatter plot:



So, we can again see that DBSCAN was able to identify the outliers and group them separately, ie. did not include them in any of the clusters, whereas KMeans failed to group outliers separately and grouped them with other core points as clearly visible from the scatter plot.