Lab Assignment 6
Prakhar Gupta
B21AI027

Question 1:

Preprocessing-

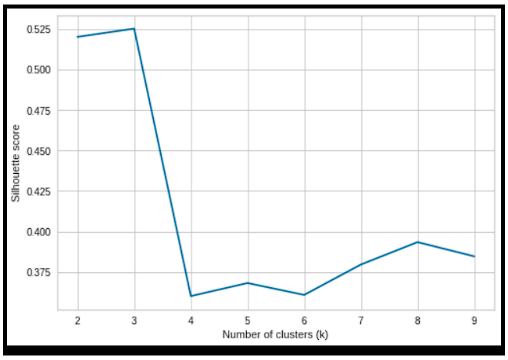
- Downloaded dataset using **wget** command called using **os.system**
- Loaded **glass.data** in df using pd.read_csv
- Gave column names as **column_names=["Id","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type of glass"]**
- Dropped **"Id"** column from df
- Checked for not filled rows using **df.isnull().sum()**
- Applied **MinMaxScalar()** to each column
- Checked **df.describe()**
- Plotted df using **pairplot**
- Converted **df to X,y**
- Applied **MinMaxScalar()**

Part A -

- Applied **KMeans** with **n_clusters=3**
- Fitted the dataset
- Plotted ScatterPlot

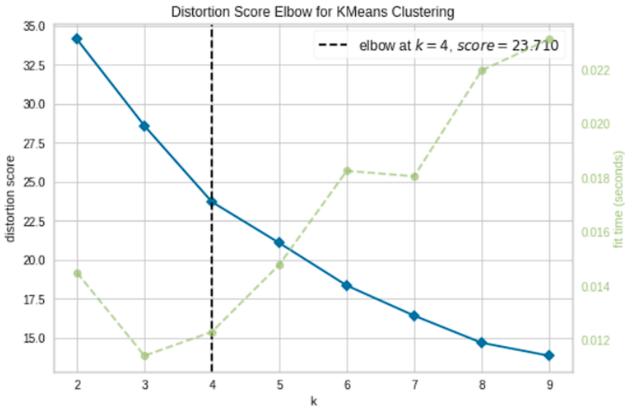
Part B-

- Found best value of k using **Silhouette Score**
- Plotted the **k vs Silhouette Score**
- ```
  Optimal value of k is 3 ,with silhouette score as 0.5252437159134774
  ```


  ```

Part C-

- Found optimal method of k using **Elbow Method**
- We used **KElbowVisualizer**



- We got the elbow at k=4

Part D-

- Applied bagging with KNeighbourClassifier as base model for n_neighbours=1,2,3

```
Accuracy score for bagging (k=1) on train: 0.9590643274853801
Accuracy score for bagging (k=1) on test: 0.7674418604651163
Accuracy score for kneighbor (k=1) on train: 1.0
Accuracy score for kneighbor (k=1) on test: 0.7906976744186046

Accuracy score for bagging (k=2) on train: 0.8947368421052632
Accuracy score for bagging (k=2) on test: 0.6976744186046512
Accuracy score for kneighbor (k=2) on train: 0.8245614035087719
Accuracy score for kneighbor (k=2) on test: 0.6744186046511628

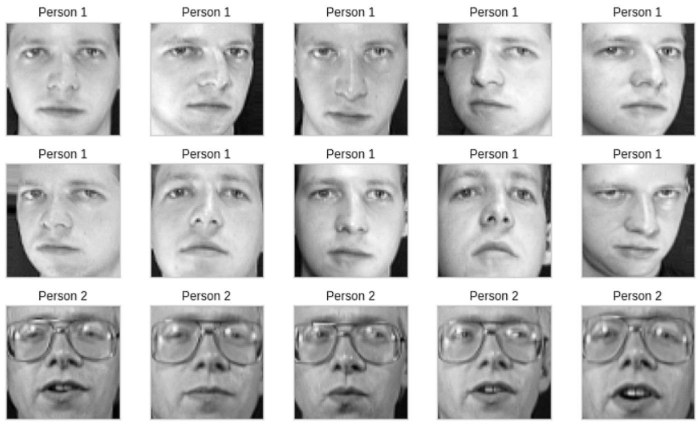
Accuracy score for bagging (k=3) on train: 0.8304093567251462
Accuracy score for bagging (k=3) on test: 0.6976744186046512
Accuracy score for kneighbor (k=3) on train: 0.7953216374269005
Accuracy score for kneighbor (k=3) on test: 0.7209302325581395
```

- We can see that knn is somewhat **overfitting** ,i.e. Giving **high variance** , so when we applied **Bagging we reduce the overfitting** as Bagging reduces overfitting , but due to this only it slightly increases the bias
- **With increase in k** the model tends to have **decrease high_variance** and and **increase low_bias**, which is why on the test set the Bagging score is almost similar or little less than the knn

<u>Question 2:</u>

Initials-

- Loaded dataset into **data**
- Converted **data to X,y**
- Printed shape of X,y
- Plotted some **images for visualisation**



Part A&B-

- Made a **scratch built KMeans** class
- The class contains **`fit,assign_clusters,closest_centroid,get_centroids,`** **`is_converge,SSE`** functions

Part C-

- Initialize **random 40 points with 4096 dimension**
- **Fitted KMeans_Scratch** on above randomly initialised centroid points
- Printed points per cluster
- `Points_per_class=[1, 17, 33, 9, 5, 21, 3, 13, 12, 10, 10, 4, 7, 14, 14, 2, 1, 10, 15, 10, 31, 9, 1, 1, 11, 3, 11, 11, 6, 2, 4, 4, 3, 1, 4, 26, 12, 13, 3, 33]`

Part D-

- Defined a function **array_to_img** to convert 1-D array to 2-D array so it can be plotted as a image
- Plotted the images considering the centroids predicted from the KMeans_Scratch

Part E-

- Plotted one image of each class as predicted by the KMeans_Scratch on the randomly initialised centroid

Part F-

- **Initialized centroid as random 40 points from the dataset**
- Printed points per cluster
- `Points_per_cluster=[9, 11, 6, 16, 20, 10, 3, 6, 14, 6, 12, 5, 17, 6, 14, 5, 5, 10, 5, 4, 11, 9, 8, 13, 16, 13, 10, 6, 5, 19, 9, 13, 10, 10, 4, 11, 5, 14, 16, 14]`

- Plotted the images considering the centroids predicted from the KMeans_Scratch

Part G-

- Plotted one image of each class as predicted by the KMeans_Scratch on centroid initialisation as image per class

Part H-

- Printed `Sum of Squared Error (SSE) for random_intialised points model and per_cluster_intialised points model`
- `Sum of Squared Error (SSE) for random_initialised points model: 3.321594695205111`
  `Sum of Squared Error (SSE) for per_cluster_initialised points model: 2.9923013387031974`
- When we initialised **centroid to be points per class** , **it is the best case** as can be seen using SSE, this thing can also be seen where we can see that points per cluster for per_cluster_initialised are all nearby 10, whereas it is spread out in random_initialised

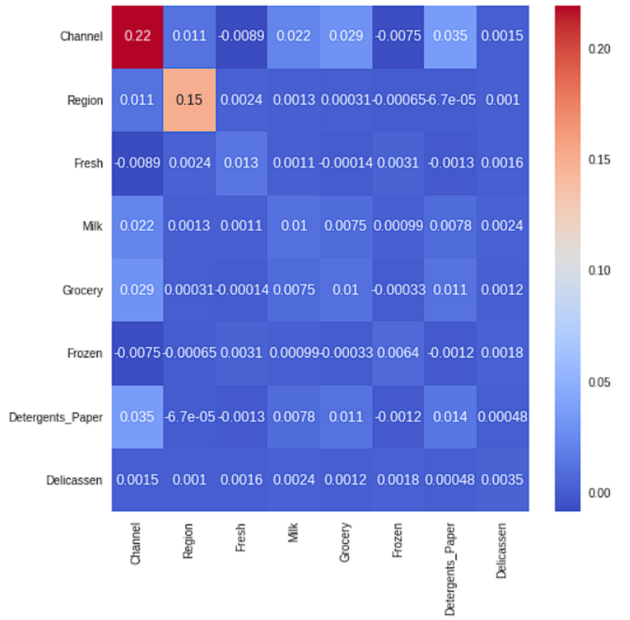
<u>Question 3:</u>

Part A-

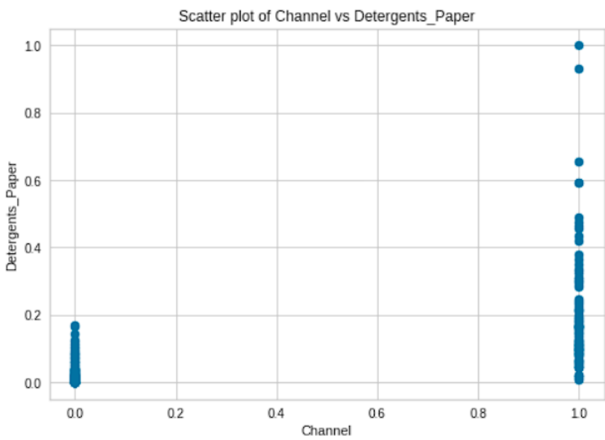
- Downloaded dataset using **wget** command called using **os.system**
- Loaded **Wholesalecustomersdata.csv** in df using pd.read_csv
- Checked for not filled rows using **df.isnull().sum()**
- Checked **df.dtypes()**
- Applied **MinMaxScalar()** to each column
- For insights about data we used **df.describe()**

Part B-

- For getting covariance matrix we used **df.cov()**
- Plotted heatmap of covariance matrix



- Using for loop over each 2 feature, find the features which has **highest covariance**(*As if an outlier is present in one then due to high covariance it should also be present in the other feature*)
- `Max_cov_matrix=[[0.21907227 0.03476772]`
- ` [0.03476772 0.01364002]]`
- `Features with max_cov : ['Channel', 'Detergents_Paper']`
- So we get max_cov for **['Channel','Detergents_Paper']** , the same result can also be seen from the heatmap
- Visualising best Feature with **highest covariance** using scatter plot



- Visualising best Feature with **highest covariance** using boxplot

Part C-

- Applied **DBSCAN**
- **Fitted** the dataset
- Plotted **pairplot**

Part D-

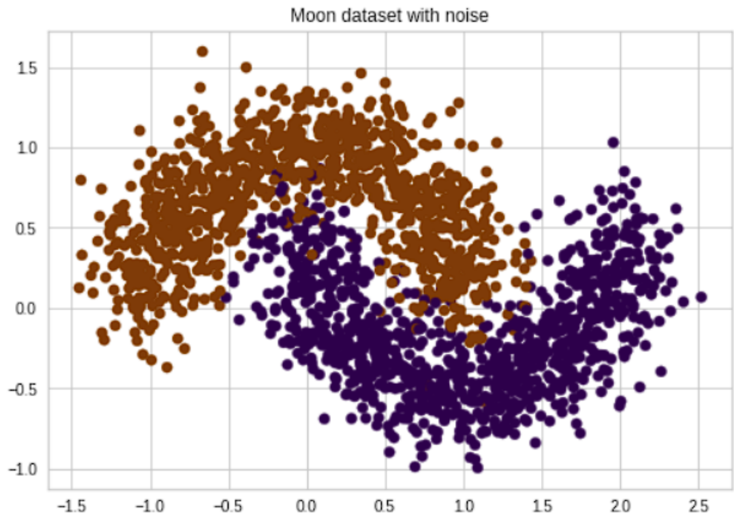
- Applied **KMeans**
- **Fitted** the dataset
- Plotted **pairplot**

```
DBSCAN Silhouette score: 0.6360287841835186
KMeans Silhouette score: 0.5928657922516165
```

- We can see that DBScan is performing better than KMeans, the same can be evident from the graph, where DBScan scatter plots has more segregated points than that of the plot of KMeans. DBScan is able to see the outliers because of which its Silhouette score is high where as KMeans is not able to see the outliers

Part E-

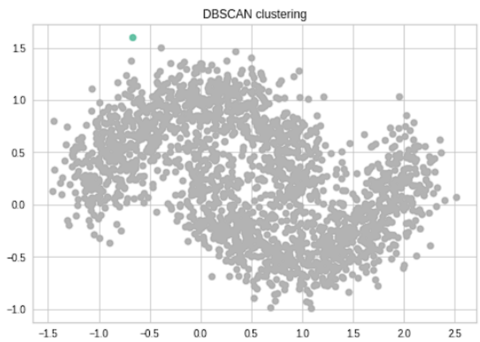
- Made **X,y** using make_moons function of scikit learn
- Created 2000 samples and also added 20% noise to the dataset
- Visualised the dataset using scatter plot

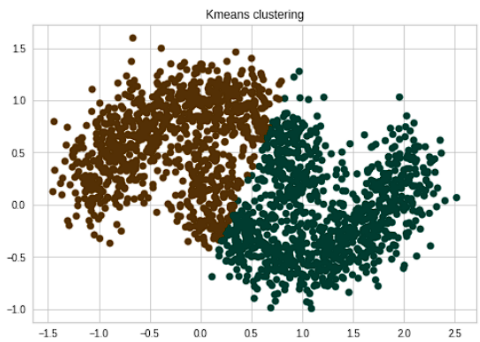


- Made **DBScan** and **KMeans** Clustering model

```
DBSCAN(eps=0.2, min_samples=5)
KMeans(n_clusters=2)
```

- Fitting these two model on the dataset
- Predicted the dataset value from these models and plotted the prediction using scatter plot

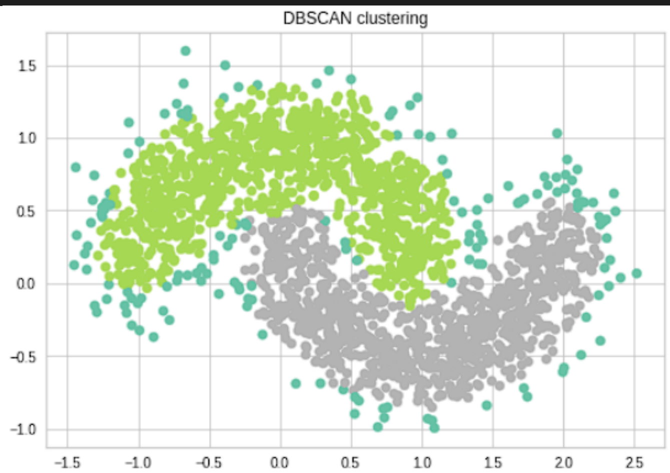




- Clearly the **KMeans is better than DBScan** as evident from the plots, the reason for DBScan to perform this much poor is that it considers the surrounding atom counts and their distance , and our dataset has noise due to which it consider almost whole dataset as one cluster.

- We can try to tune DBScan by changing the min_samples and epsilon, then we can get a good result , where we get 3 classes where one is predicted as outlier

```
DBSCAN(eps=0.2, min_samples=50)
```



- In the tuned version although it suggests 3 classes but its predictions are **better than that of KMeans** because KMeans is not suitable for non-linear data