Lab 6 Report

Nakkina Vinay (B21AI023)

Question 1

- Using the head() printing the dataset

	Id.no:	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type of glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

- Using the info() function

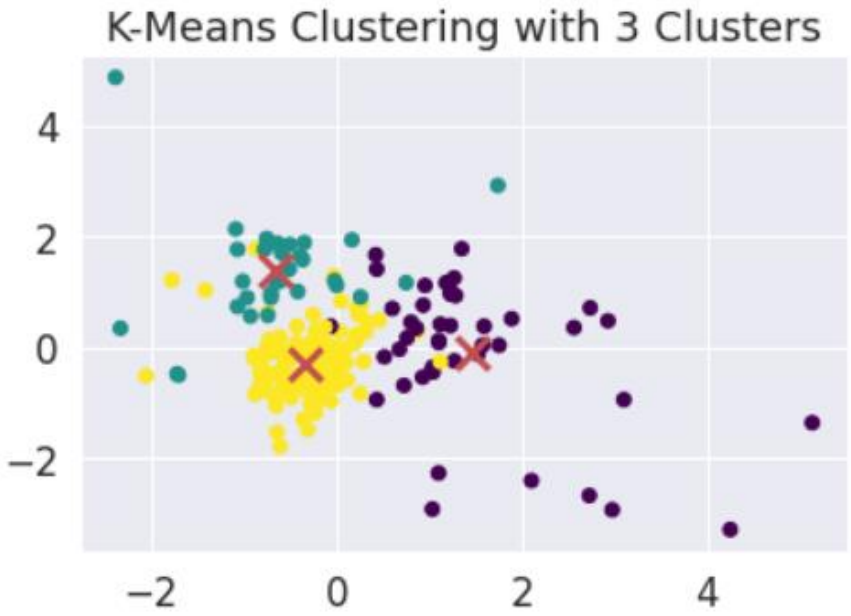
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id.no:         214 non-null    int64
 1   RI             214 non-null    float64
 2   Na             214 non-null    float64
 3   Mg             214 non-null    float64
 4   Al             214 non-null    float64
 5   Si             214 non-null    float64
 6   K              214 non-null    float64
 7   Ca             214 non-null    float64
 8   Ba             214 non-null    float64
 9   Fe             214 non-null    float64
 10  Type of glass  214 non-null    int64
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

- Dropping the column Id.no: since it is not required
- The dataset was analysed and it was a complete dataset after dropping and no pre-processing is required

Subpart 1:

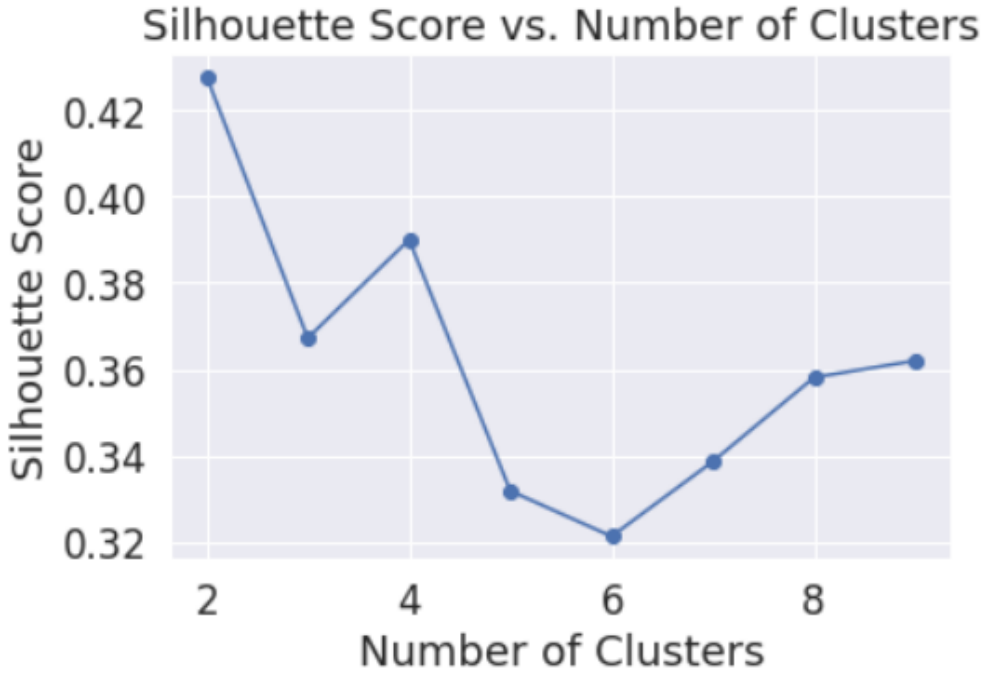
- Building a K-Means clustering algorithm
- First removing the target variable column i.e Type of glass column
- Scaling the data
- Defining the number of clusters as three
- Fitting the k-means model to the data

- Getting the labels and centroids for each cluster
- Plotting the datapoints and the centroids



Subpart 2:

- Defining the range of clusters to evaluate
- Iterating over the defined range of clusters and calculating the Silhouette score for each value of k
- Plotting the Silhouette scores as a function of k



- The optimal value of k is the one that corresponds to the highest Silhouette score. In this case, we can see that the optimal value of k is 2, as this value have the highest Silhouette score.

- A high Silhouette score indicates that the clustering is well-defined and the clusters are well-separated.

Subpart 3:

- Using the Elbow method to find the optimal value of k
- Iterating over the range of clusters and calculating the sum of squared distances for each value of k
- Plotting the sum of squared distances as a function of k

Subpart 4:

- First splitting the data into training and testing sets
- Defining the base model as KNeighboursClassifier
- Defining the number of estimators as 10
- Defining the different values of k aas 1,2,3 to use for the KNN classifier
- Iterating over the different k values and fitting the bagging model
- Calculating the accuracy on the test data for each value of k

```
Accuracy with K=1: 0.57
Accuracy with K=2: 0.60
Accuracy with K=3: 0.60
```

- The accuracy increases as we increase the value of K, up to a certain point. This is because increasing the value of K reduces the variance of the K-nearest neighbors

classifier, making it less susceptible to overfitting. However, increasing the value of K too much can increase the bias of the classifier, causing it to underfit the data.

- By using bagging with the K-nearest neighbors classifier, we can reduce the variance of the classifier, which can lead to a more stable and accurate model.
- We can observe that the accuracy of the bagging model generally increases as we increase the value of K. This suggests that the variance of the K-nearest neighbors classifier is being reduced by bagging, leading to more accurate predictions.

Question 2

- Fetching the dataset from sklearn library and printing some sample of images

Subpart 1 and 2:

- Implementing k-means clustering algorithm from scratch
- KMeans class takes three parameters
    - Value of k
    - Maximum iterations
    - Initial cluster center points = init_centers
- KMeans class consists of

- __init__ for initialisation of different variables being used in the class
- Fit()
    - This function first initialises centroids
    - Iterates until convergence or maximum iterations and assigns labels to datapoints based on centroids and updates centroids
    - Checks if they are converged or not
    - Computes SSE for the initialization
- Predict()

Subpart 3:

- Train the k-means model on the dataset with k = 40 and 40 random 4096-dimensional points (in input range) as initializations.
- Printing the number of points in each cluster

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

Subpart 4:

- Visualize the cluster centers of each cluster as 2-d images of all clusters.

Subpart 5:

- Visualising 10 images corresponding to each cluster.

- Like this 10 images for each cluster are printed only some are attached above

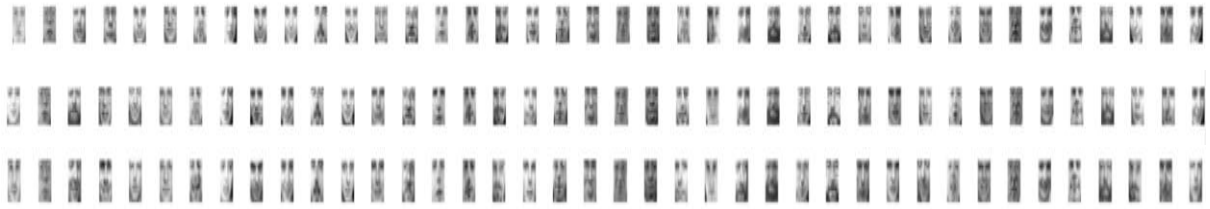
Subpart 6:

- Training another k-means model with 1 images from each class as initializations and reporting the number of points in each cluster and visualising the cluster centers.

Number of points in each cluster: [14, 10, 13, 11, 23, 10, 7, 5, 13, 3, 15, 6, 11, 10, 5, 2, 4, 16, 5, 4, 5, 10, 21, 17, 10, 5, 10, 10, 9, 19, 14, 10, 10, 10, 6, 2, 5, 21, 10, 9]

Subpart 7:

- Visualising 10 images corresponding to each cluster.



- Like this 10 images for each cluster are printed only some are attached above

Subpart 8:

- Evaluating Clusters of part 3 and part 6 with Sum of Squared Error (SSE) method.

```
SSE for Model 1: 2802.9814
SSE for Model 2: 2706.278
```

- A lower SSE score indicates a better clustering, as it means that the data points are closer to their assigned cluster centers.

Question 3

Subpart 1:

- Getting the dataset and printing it

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

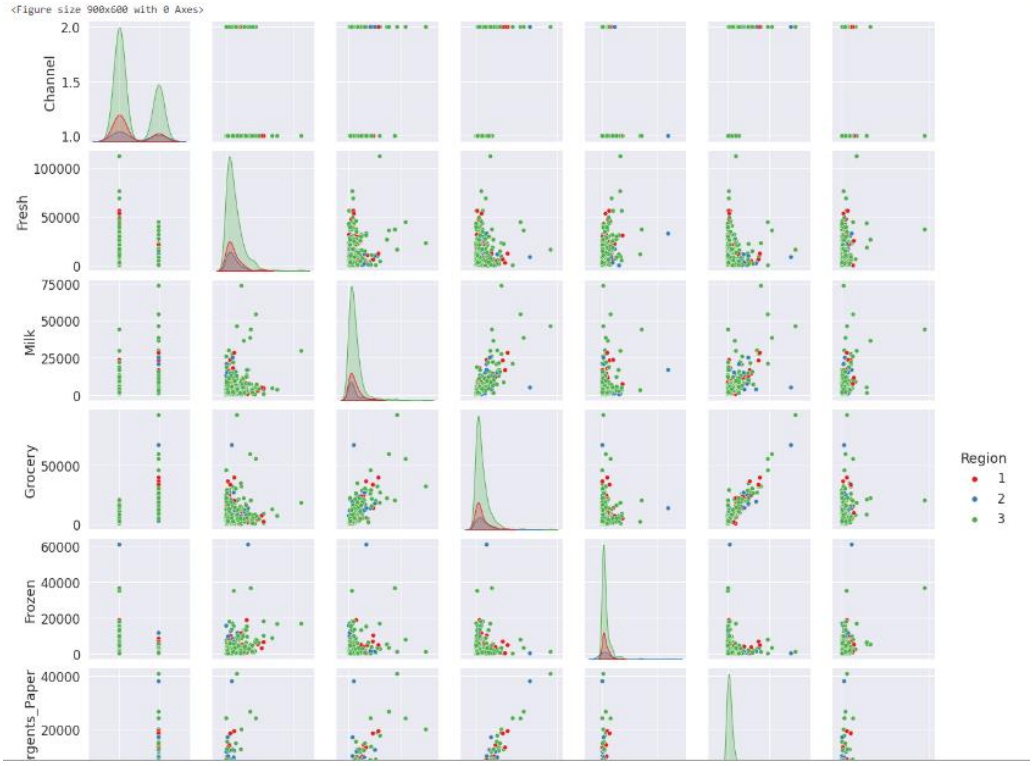
- Getting the info of the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

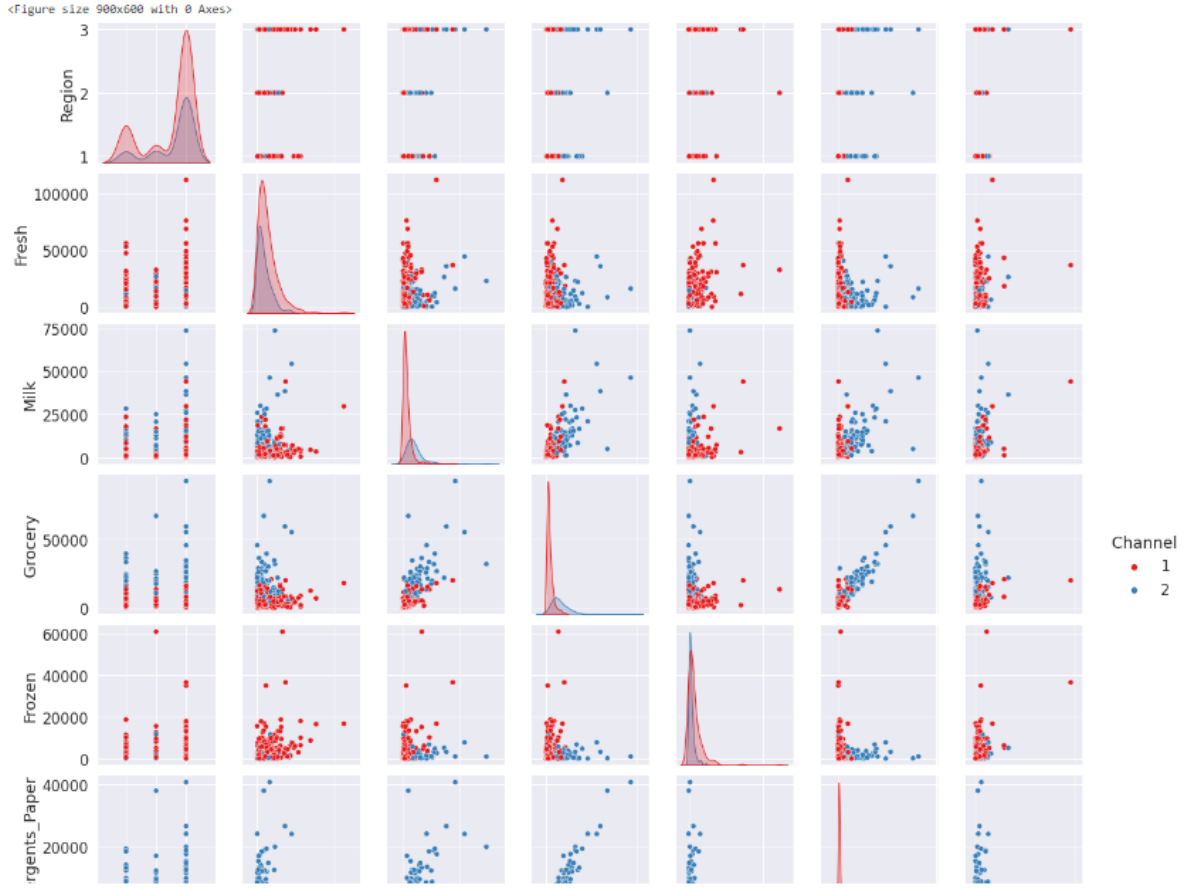
- Using the describe() on the dataset for more information about the dataset

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

- There are no empty values in the dataset
- Visualisation using the pairplot with hue=Region



- Visualisation using the pairplot with hue=Channel



- Dropping the Region and Channel columns
- Standardizing the data using StandardScaler and calculating the mean and standard deviation

```
(440, 6)

Mean of standarised data:
 [-3.43159844e-17  0.00000000e+00 -4.03717464e-17  3.63345717e-17
  2.42230478e-17 -8.07434927e-18]


Standard deviation of standarised data:
 [1. 1. 1. 1. 1. 1.]
```

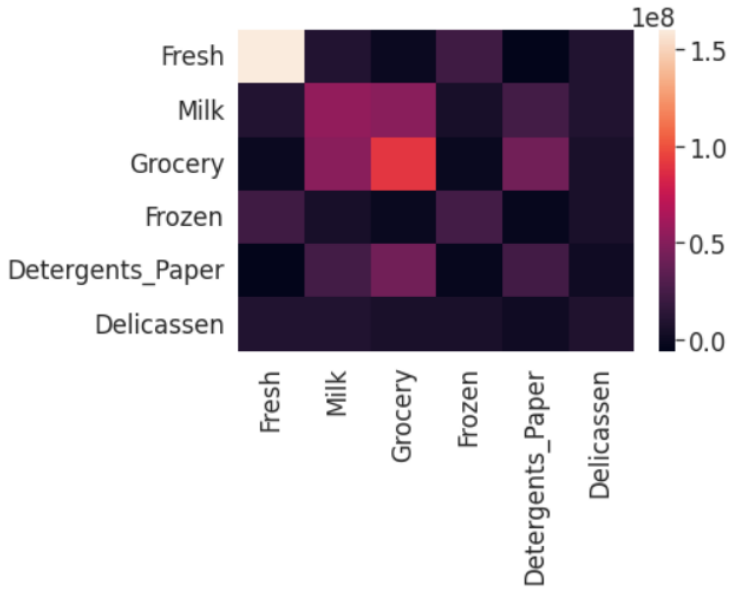
Subpart 2:

- Computing the covariance matrix and visualising the covarince matrix using the heapmap

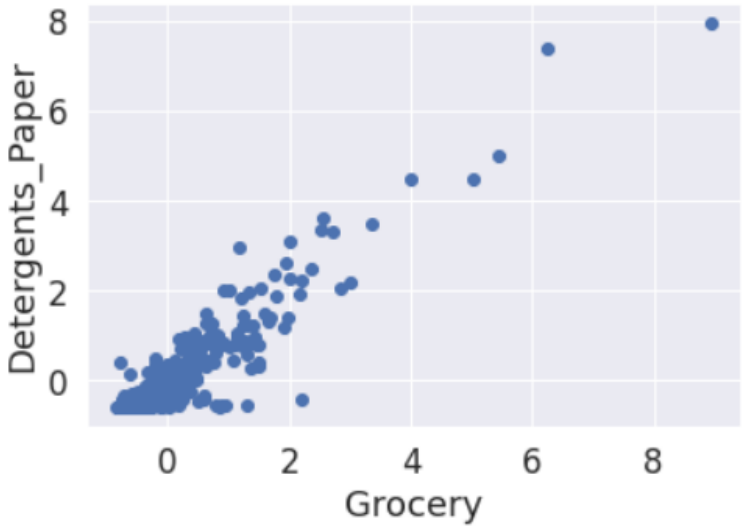
	Fresh	Milk	Grocery	Frozen	\
Fresh	1.599549e+08	9.381789e+06	-1.424713e+06	2.123665e+07	
Milk	9.381789e+06	5.446997e+07	5.108319e+07	4.442612e+06	
Grocery	-1.424713e+06	5.108319e+07	9.031010e+07	-1.854282e+06	
Frozen	2.123665e+07	4.442612e+06	-1.854282e+06	2.356785e+07	
Detergents_Paper	-6.147826e+06	2.328834e+07	4.189519e+07	-3.044325e+06	
Delicassen	8.727310e+06	8.457925e+06	5.507291e+06	5.352342e+06	

	Detergents_Paper	Delicassen
Fresh	-6.147826e+06	8.727310e+06
Milk	2.328834e+07	8.457925e+06
Grocery	4.189519e+07	5.507291e+06
Frozen	-3.044325e+06	5.352342e+06
Detergents_Paper	2.273244e+07	9.316807e+05
Delicassen	9.316807e+05	7.952997e+06

<AxesSubplot:>

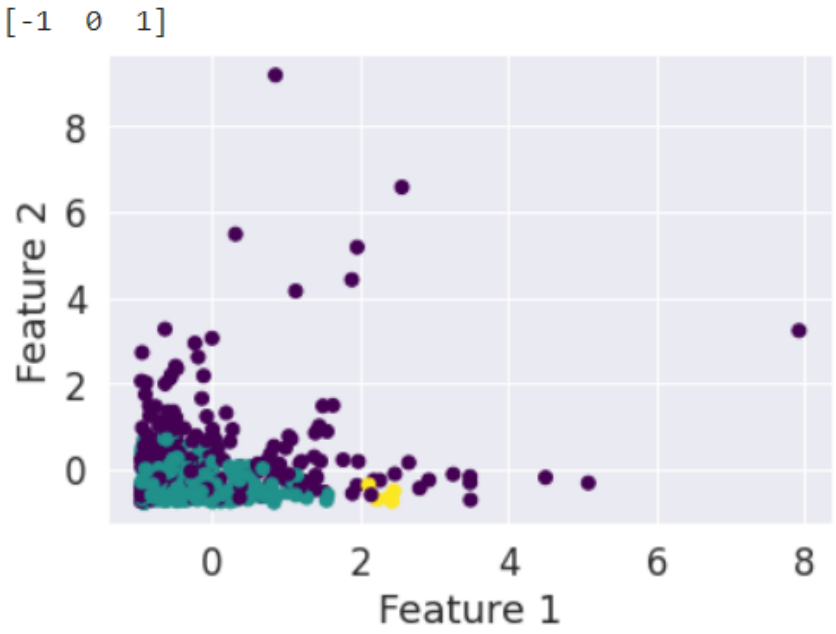


- The resulting heatmap shows that the pair of features with the highest covariance is "Grocery" and "Detergents_Paper". Therefore, we can best visualize the outliers by plotting a scatter plot of these two features.
- Converting the dataset to dataframe and plotting the scatter plot of the pair of features



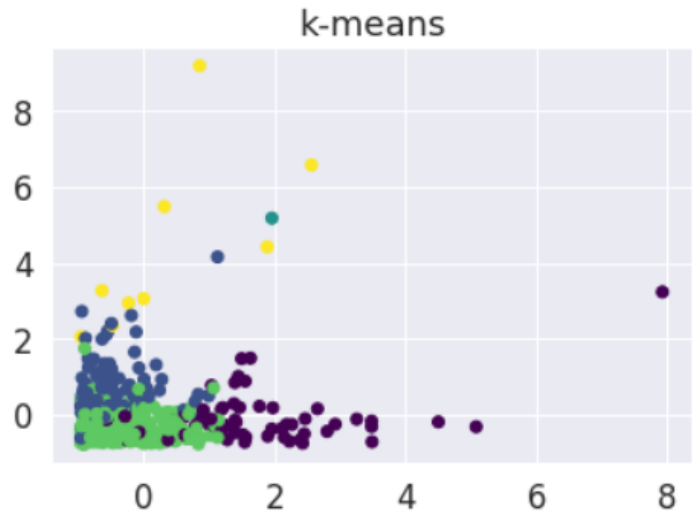
Subpart 3:

- Applying the DBSCAN to cluster the data and visualising the same



Subpart 4:

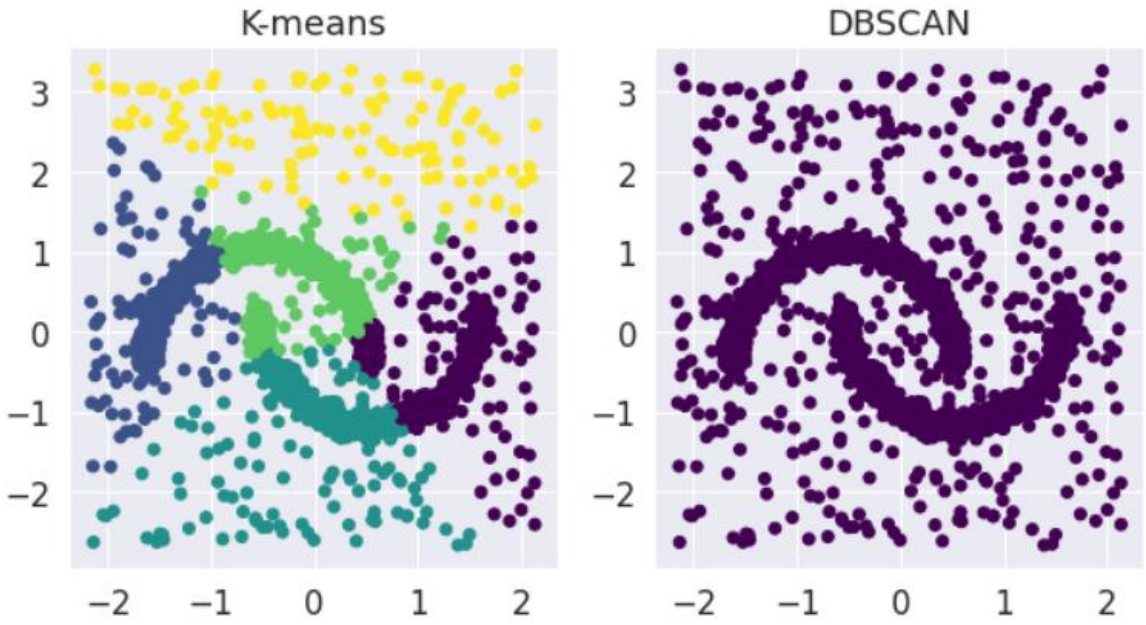
- Applying K-Means clustering on the dataset and visualising the same



- The k-means and DBSCAN algorithms produce quite different clusterings of the data. The k-means clustering produces circular clusters, while the DBSCAN clustering produces more irregularly shaped clusters.
- Overall, the choice of clustering algorithm will depend on the specific characteristics of the dataset and the goals of the analysis. In this case, it's not immediately clear which algorithm is better, as they both produce somewhat different clusterings.

Subpart 5:

- Generating the dataset using make_moons function with n_samples=2000
- Adding some additional noise
- Here, we're using a random mask to select 20% of the data points and then adding some random noise to their positions.
- Scaling the data and applying DBSCAN to cluster the data
- Also applying the K-means clustering on the dataset
- Visualising the k-means and DBSCAN clusters



- Both clustering algorithms are able to identify the two moon-shaped clusters in the data, but they handle the noise differently.
- K-means clustering produces circular clusters that are distorted by the presence of the noise, while DBSCAN clustering produces irregularly shaped clusters that are less affected by the noise.
- In this case, DBSCAN clustering might be considered the better algorithm since it produces clusters that more closely match the true shape of the underlying data.
- However, the choice of clustering algorithm will depend on the specific characteristics of the dataset and the goals of the analysis.