Datasets

The dataset selected for this study is the wholesale customer dataset on the UCI machine learning repository. The source for this dataset is “Margarida G. M. S. Cardoso, margarida.cardoso@iscte.pt, ISCTE-IUL, Lisbon, Portugal” (UCI Machine Learning Repository, 2011).

Explanation and preparation of datasets

The dataset downloaded for this study is a dataset that contains the annual spendings on different categories of products for 440 customers. The dataset contains 441 rows and 8 columns. The rows contain the header and the observations for the 440 customers. We’d be giving brief explanations of the 8 variables as follow.

“1) FRESH: annual expenditure monetary units (m.u.) on fresh products

2) MILK: annual expenditure (m.u.) on milk products

3) GROCERY: annual expenditure (m.u.) on grocery products

4) FROZEN: annual expenditure (m.u.) on frozen products

5) DETERGENTS_PAPER: annual expenditure (m.u.) on detergents and paper products

6) DELICATESSEN: annual expenditure (m.u.) on and delicatessen products

7) CHANNEL: client’s channel type - Horeca channel (Hotel/Restaurant/Cafe) or Retail channel

8) REGION: client’s region (Lisnon, Oporto or Other) “ (UCI Machine Learning Repository, 2011).

We’d be looking at the description of the dataset in the following images.

CLUSTERING ALGORITHMS

In [1]:
```
#importing important libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Set number of threads
import os
os.environ["OMP_NUM_THREADS"] = '1'
```

In [2]:
```
#loading the dataset
dataset=pd.read_csv('Wholesale customers data.csv')
```

In [3]:
```
dataset.shape
```

Out[3]: (440, 8)

In [4]:
```
dataset.head()
```

Out[4]:

	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

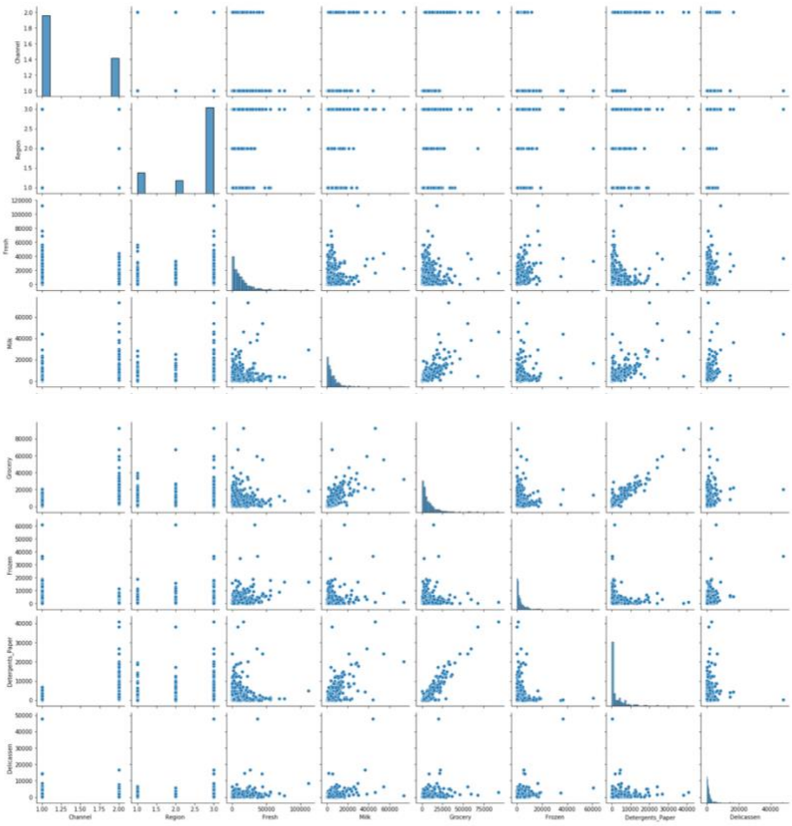
```
In [5]: #Description of the dataset
        dataset.describe(include = "all")
```

Out[5]:

	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

We also created a plot for the visualisation of the variables.

```
In [14]: #Visualisation of the variables
         sns.pairplot(dataset.iloc[:,0:8])

Out[14]: <seaborn.axisgrid.PairGrid at 0x7fcfa7064210>
```



We checked for missing values and from the image below, we saw that we didn't have any missing values.

```
In [10]: #checking for missing values
         dataset.isnull().sum()

Out[10]: Channel             0
         Region              0
         Fresh               0
         Milk                0
         Grocery             0
         Frozen              0
         Detergents_Paper    0
         Delicassen          0
         dtype: int64
```

We can see that Channel and Region are categorical variables from the definitions given above. Next, we see the count for these variables, and we replace the numbers with the category values. The values were gotten from the metadata provided on the website.

```
In [13]: #exploring the unique values in the categorical features
         print("Total categories in the feature Region:\n", dataset["Region"].value_counts(), "\n")
         print("Total categories in the feature Channel:\n", dataset["Channel"].value_counts())

         Total categories in the feature Region:
          3    316
         1     77
         2     47
         Name: Region, dtype: int64

         Total categories in the feature Channel:
          1    298
         2    142
         Name: Channel, dtype: int64
```

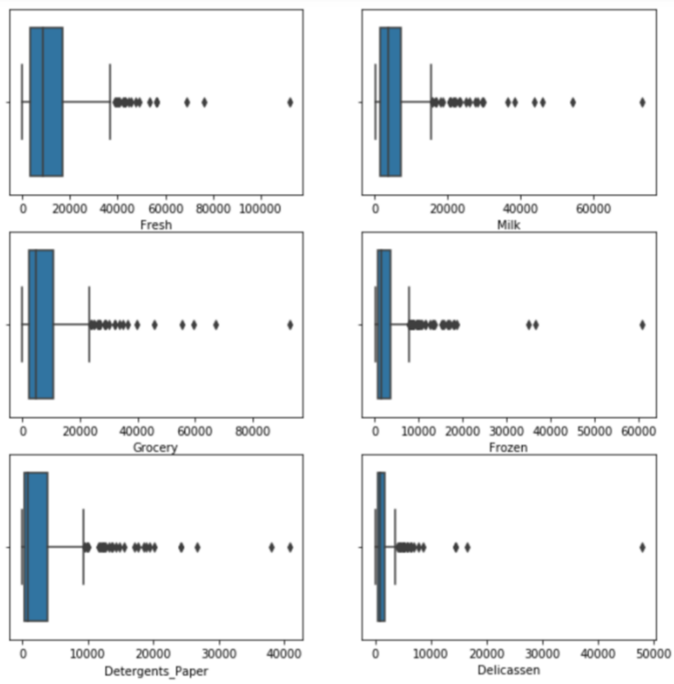
```
In [15]: #Changing values of the categorical variables

         dataset['Channel']=dataset['Channel'].replace([1,2],['Horeca','Retail'])
         dataset['Region']=dataset['Region'].replace([1,2,3],['Lisbon','Oporto', 'Other'])
```

```
In [16]: dataset.head()
```

Out[16]:

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

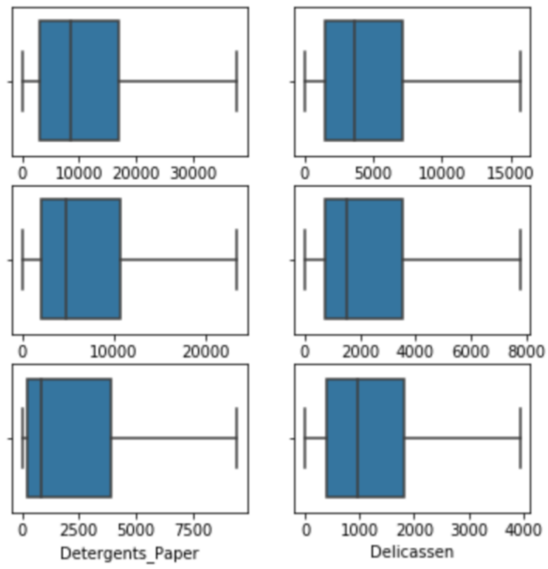
Outliers are objects that do not belong to any clusters. K-means algorithm is influenced by outliers (S Joel Franklin, 2019), so we check for outliers in the variables to see if we would find and then we deal with outliers if any is found. We would be using the data with outliers in the DBSCAN algorithm.



Now, we will be treating the outliers for the k-means algorithm. For treating the outliers, we replaced them with their Inner fences.

```
In [12]: #Treating outliers
DataKM = dataset
DataKM = DataKM.drop('Region', inplace=False, axis=1)
DataKM = DataKM.drop('Channel', inplace=False, axis=1)

# replacing the outliers with their Inner fences
for k in list(DataKM.columns):
    IQR = np.percentile(DataKM[k],75) - np.percentile(DataKM[k],25)

    Outlier_top = np.percentile(DataKM[k],75) + 1.5*IQR
    Outlier_bottom = np.percentile(DataKM[k],25) - 1.5*IQR

    DataKM[k] = np.where(DataKM[k] > Outlier_top,Outlier_top,DataKM[k])
    DataKM[k] = np.where(DataKM[k] < Outlier_bottom,Outlier_bottom,DataKM[k])
```

```
In [13]: #Boxplot for Numerical Values
fig, ax = plt.subplots(3,2,figsize= (6,6))
sns.boxplot(x=DataKM['Fresh'], ax=ax[0,0])
sns.boxplot(x=DataKM['Milk'], ax=ax[0,1])
sns.boxplot(x=DataKM['Grocery'], ax=ax[1,0])
sns.boxplot(x=DataKM['Frozen'], ax=ax[1,1])
sns.boxplot(x=DataKM['Detergents_Paper'], ax=ax[2,0])
sns.boxplot(x=DataKM['Delicassen'], ax=ax[2,1])

Out[13]: &lt;matplotlib.axes._subplots.AxesSubplot at 0x7fa2d859c350&gt;
```



Now we would be scaling both datasets to meet a standard normal distribution. ‘Dataset’ on our worksheet will be used for the DBSCAN algorithm and ‘DataKM’ on our worksheet will be used for the k-means algorithm.

```
In [15]: #Standardisation
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

#Creating dummies for categorical varibles
dataset = pd.get_dummies(dataset,columns=['Channel','Region'],drop_first=True)
DataKM = pd.get_dummies(DataKM,columns=['Channel','Region'],drop_first=True)


dataset=sc.fit_transform(dataset)
DataKM=sc.fit_transform(DataKM)
```

Implementation in Python

Now that we have our dataset ready, we will be using two different clustering algorithms which are K-Means algorithm, and DBSCAN algorithm to create clusters.

K-MEANS ALGORITHM

K-Means clustering is one of the Partitioning clustering algorithms. It is a popular and efficient unsupervised machine learning algorithm. It divides objects into clusters. Objects that share similarities are put in the same cluster and objects that are dissimilar are put into other clusters. K is the number of clusters that will be created. The number K is to be defined by the user (Mayank Banoula, 2022). One may wonder how to know the best number of clusters to choose. Though we can guess the different clusters that can be created from the type of dataset that we are working with, there is a way of finding out the optimal K for a dataset. The way the algorithm works is described in the steps below.

- Step 1: Find out the number K.
- Step 2: Choose K different centroids (cluster initialisation) at random.
- Step 3: Each point's distance is measured from the centroid.
- Step 4: Assign each point to its closest cluster.
- Step 5: A new centroid is calculated by find the mean of its data.
- Step 6: Step 3 – 5 is repeated with the new centroid
- Repeat until the there's a convergence, i.e., there's no significant difference in the newly calculated centroids or we reach the maximum number of iterations (Arif R, 2020).

K-means algorithm has various distance metrics which measures the similarity between two data points. In this study, we will be making use of the Euclidean distance measure because it is widely used with K-means algorithm.

$$distance(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

To choose the optimal value of K, we will be using the elbow method. It selects a range of values and picks the best one. It computes the average distance and the sum of the squares of the points. The point at which the value of K declines the most on the plot (the cost function's value as created by various K values) is called the elbow. We are calculating WCSS (Within-Cluster Sum of Square) for each value of K.

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

$$x_i = data\ point\ \&\ c_i = closest\ point\ to\ centroid$$

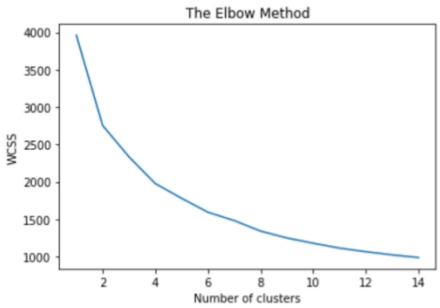
We used this as our first algorithm because of its popularity as a clustering algorithm. We also wanted to study how the clusters would be formed with an algorithm that requires treatment of outliers so we can compare with one that doesn't require this.

Creating the Model in Python

We first used the elbow method to find the optimal K. From the image below, we picked our optimal K to be 5.

K - Means Algorithm

```
In [16]: #finding the optimal value of K
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(DataKM)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 15), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



We were able to fit the K-Means algorithm to the dataset as seen below.

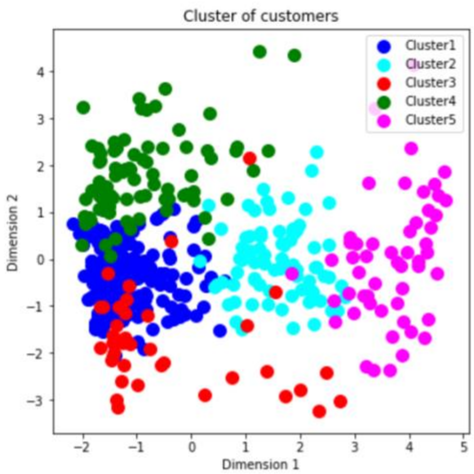
```
In [40]: #Fitting K-Means to the dataset
model = KMeans(n_clusters=5, init = 'k-means++', random_state = 0)
y_kmmodel = model.fit_predict(DataKM)
```

Because of the dimensions of the data set, we are unable to plot a scatter plot that would show us the 5 clusters that we got from the algorithm. To be able to visually view these clusters, we would be using a dimension reduction technique called PCA. The strongest trends in a dataset or between groups in a dataset are typically visualised using PCA. We would be reducing the dimension of our data from 8 dimensions to 2 dimensions which will then enable us to plot a scatter plot.

```
In [18]: #Principal component analysis to get the first 2 Principle components
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pcaKM = pca.fit_transform(DataKM)
pca.explained_variance_ratio_

Out[18]: array([0.37811059, 0.18537554])
```

We can now visually see the scatter plots for the 5 clusters that were created.

```
In [42]: #Visualising the clusters

colors = ['blue', 'cyan', 'red', 'green', 'magenta']
plt.figure(figsize=(6,6))
for i in range(5):
    plt.scatter(pcaKM[y_kmmodel== i, 0], pcaKM[y_kmmodel== i, 1],
                s = 100, c = colors[i], label = 'Cluster' +str(i+1))
plt.title('Cluster of customers')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```



We would be using the DBSCAN algorithm on the data with outliers to see how it creates clusters.

DBSCAN (Density Based Spatial Clustering of Applications with Noise) ALGORITHM

DBSCAN algorithm is an unsupervised machine learning algorithm. This algorithm was proposed by Martin Ester et al. in 1996 (Abhishek Sharma, 2020). It is a density-based clustering algorithm that works based on the intuition that clusters are dense region of data points separated by regions of lower density. It is based on the notion of noise and clusters. Even with noise, density-based clustering can be beneficial for arbitrary forms. Data points that are 'densely grouped' are grouped together in one cluster. One thing that makes this algorithm very interesting is how it is insensitive to outliers. Also, we do not need to know the number of clusters beforehand like the K-Means algorithm.

This algorithm has two parameters which are epsilon (Eps) and minimum of points necessary to produce a dense region (MinPoints) (Shritam Kumar Mund, 2019).

Eps is a distance measure that defines how data points should be close to each other to form a neighbourhood. The chosen value for epsilon cannot be too large or small, if not, it will affect the clusters built. The k-distance graph is used to find the suitable epsilon value.

MinPoints is the minimum of points necessary to produce a dense region.

After clustering, we observe three different points. Core Point, Border and Noise.

The way the algorithm works is described in the steps below.

- Step 1: Select a data point randomly and find all the neighbours withing eps, then identify the core point.
- Step 2: Create a new cluster for any core point not assigned to a cluster.
- Step 3: Find and allocate all points that are recursively related to the core point cluster.
- Step 4: Reiterate through points that aren't assigned and assign them to a neighbour at epsilon distance. Points not assigned to any cluster are noise (Stanley Juma, 2021).

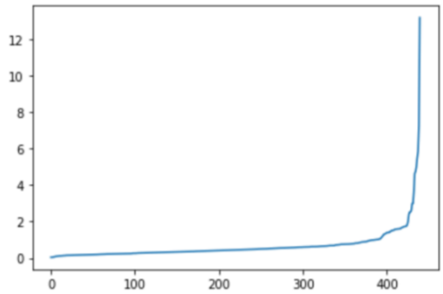
We used this method because of its ability to form clusters based on different densities, and it being robust to outliers.

Creating the Model in Python

First, we will need to find the value of epsilon.

DBSCAN ALGORITHM

In [26]:
```
#Finding the value of Epsilon

from sklearn.neighbors import NearestNeighbors
neighbours = NearestNeighbors(n_neighbors= 2)
distances, indices = neighbours.fit(dataset).kneighbors(dataset)

distances = np.sort(distances, axis = 0)
distances = distances[:, 1]
plt.plot(distances)
plt.show()
```

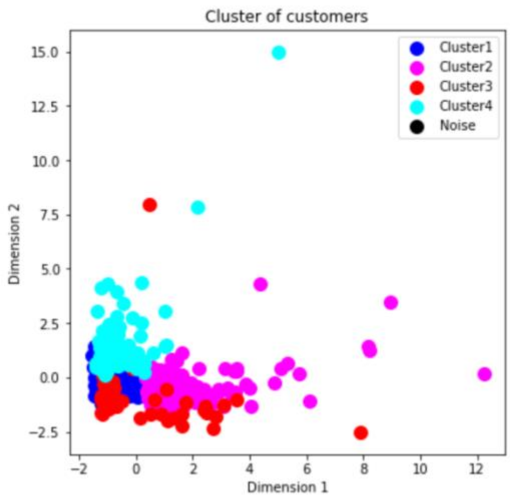


We chose an Eps of 2 since the maximum curve curvature shown in the preceding plot is about two. For the value of MinPoints, our data has more than 2 dimensions, so we chose MinPoints = 2*dim, where dim= the dimensions of our dataset (Sander et al, 1998). So, we will pick MinPoints = 16.

```
In [37]: #Implementing the DBSCAN Algorithm
         from sklearn.cluster import DBSCAN
         dbscan = DBSCAN(eps = 2, min_samples = 16)
         y_dbscan = dbscan.fit_predict(dataset)

         y_dbscan

Out[37]: array([ 0,  0,  0,  1,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  0,
                 1,  0,  1,  0,  1,  1, -1,  0,  0,  1,  1,  0,  1,  1,  1,  1,  1,
                 1,  0,  1,  0,  0,  1,  1,  1,  0,  0,  0,  0,  0, -1,  0,  0,  1,
                 1,  0,  0,  1,  1, -1,  0,  1,  1,  0, -1,  0,  0,  1, -1,  1,  0,
                 1,  1,  1, -1,  1,  0,  0,  1,  1,  0,  1,  1,  1,  0,  0,  1,  0,
                -1, -1, -1,  1,  1,  1,  1, -1, -1,  0,  1,  0,  1,  1,  1,  0,  0,
                 0, -1,  1,  1,  0,  0,  0,  0,  1,  0,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  0,  1, -1,  1,  0,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  0,  0,  1,  0,  0,  0,  1,  1,  0,  0,  0,  0,  1,  1,  1,
                 0, -1,  1,  0,  1,  0,  1,  1,  1,  1,  1, -1,  1, -1,  1,  1,  1,
                 1,  0,  0,  1,  1,  1,  0,  1,  1, -1, -1,  2,  2, -1, -1,  2,  2,
                 2, -1,  2, -1,  2, -1,  2, -1,  2,  2, -1,  2, -1,  2, -1,  2,  2,
                 2,  2, -1,  2,  2, -1,  2,  2,  2, -1,  2,  2,  2,  2,  2,  2,  2,
                 2,  2,  2,  2,  2,  2,  2, -1,  2,  2,  2,  2,  2, -1,  2,  2,  2,
                 2,  2,  2, -1,  2,  2,  2,  2,  2, -1, -1, -1,  2, -1,  2,  2,  2,
                 2,  1,  1,  1,  1,  1,  1,  0,  1,  0,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1, -1,  3, -1,  3, -1, -1,  3, -1, -1, -1, -1, -1, -1,
                -1,  3,  3, -1,  3,  3, -1,  3,  3, -1,  3,  3,  3, -1,  3,  3,  3,
                 3,  3, -1,  3,  3,  3,  3,  3, -1,  3, -1, -1, -1,  3,  3,  3,  3,
                 0,  0,  1,  0,  1,  1,  0,  0,  1,  0,  1,  0,  1,  0,  1,  1,  1,
                 0,  1,  1,  1,  1,  1,  1,  1,  0,  1,  1,  1,  1,  0,  1,  1,  0,
                 1,  1,  0,  1,  1,  0,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  0,
                 0,  1,  1,  1,  1,  1,  1,  0,  0,  1,  0,  1,  1,  0,  1,  0,  0,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  0,  1,  1])
```

From the image above, we can see that we have 4 clusters. -1 represents noise. Now, we will draw a plot to get a visual representation.

```
In [39]: #Visualising the clusters

         colors = ['blue', 'magenta', 'red', 'cyan']
         plt.figure(figsize=(6,6))
         for i in range(4):
             plt.scatter(pcaDB[y_kmmodel== i, 0], pcaDB[y_kmmodel== i, 1],
                         s = 100, c = colors[i], label = 'Cluster' +str(i+1))
         plt.scatter(pcaDB[y_kmmodel== -1, 0], pcaDB[y_kmmodel== -1, 1], s = 100, c = 'black',
         plt.title('Cluster of customers')
         plt.xlabel('Dimension 1')
         plt.ylabel('Dimension 2')
         plt.legend()
         plt.show()
```



Results analysis and discussion

Looking at the plots for both algorithms, we can see that the clusters are distinct for each algorithm. This can be because of the treatment of outliers. We chose to treat these outliers because in our study, we wanted to see how K-Means algorithm functions with removal of outliers. A study also showed that clusters gotten using an algorithm will defer to the clusters formed using another algorithm (Aravind CR, 2022).

Conclusions

We were able to see that both methods are good for clustering in customer segmentation. We were also able to establish that DBSCAN is an algorithm that is not affected by noise and K-Means is sensitive to noise.