Course name: **Python programming and analytics by Rahul Sir**

Topic name: **topic 27**

Video name: **python by Rahul sir k-means clustering**

Video length: **1 hour 22 minutes 56 seconds**

If you are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, you will use the k-Means algorithm; an unsupervised learning algorithm.

The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the Euclidean distance as measurement.

K-means clustering is a clustering algorithm that aims to partition nn observations into kk clusters.

There are 3 steps:

- Initialization – K initial "means" (centroids) are generated at random
- Assignment – K clusters are created by associating each observation with the nearest centroid
- Update – The centroid of the clusters becomes the new mean

Assignment and Update are repeated iteratively until convergence

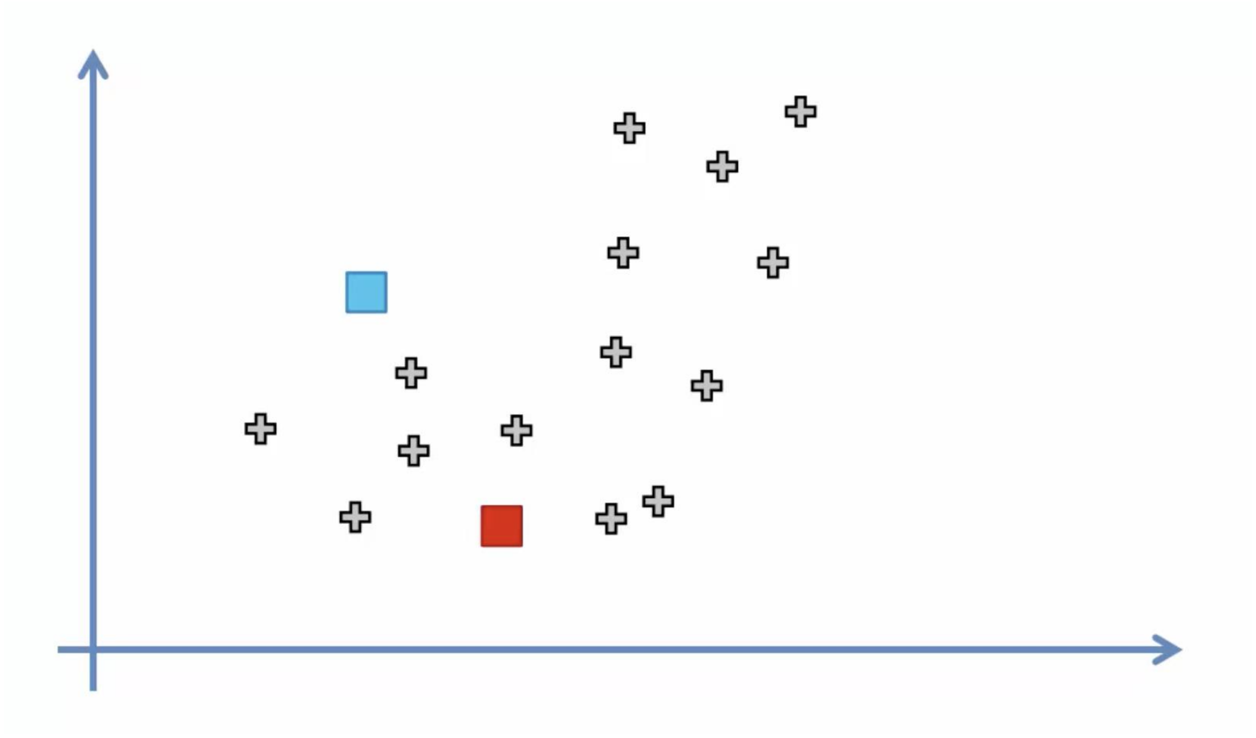
K-Means Clustering is an unsupervised machine learning algorithm. In contrast to traditional supervised machine learning algorithms, K-Means attempts to classify data without having first been trained with labeled data. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the most relevant group.

The real-world applications of K-Means include:

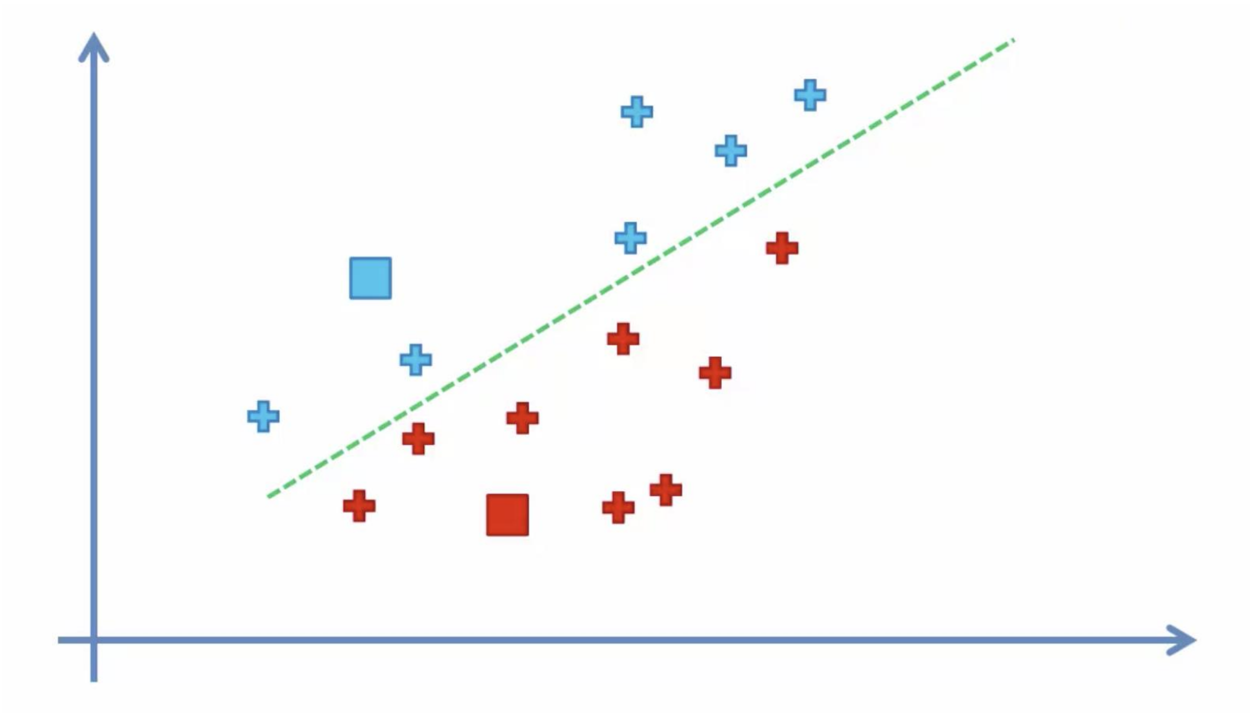
- customer profiling
- market segmentation
- computer vision
- search engines
- astronomy

How it works

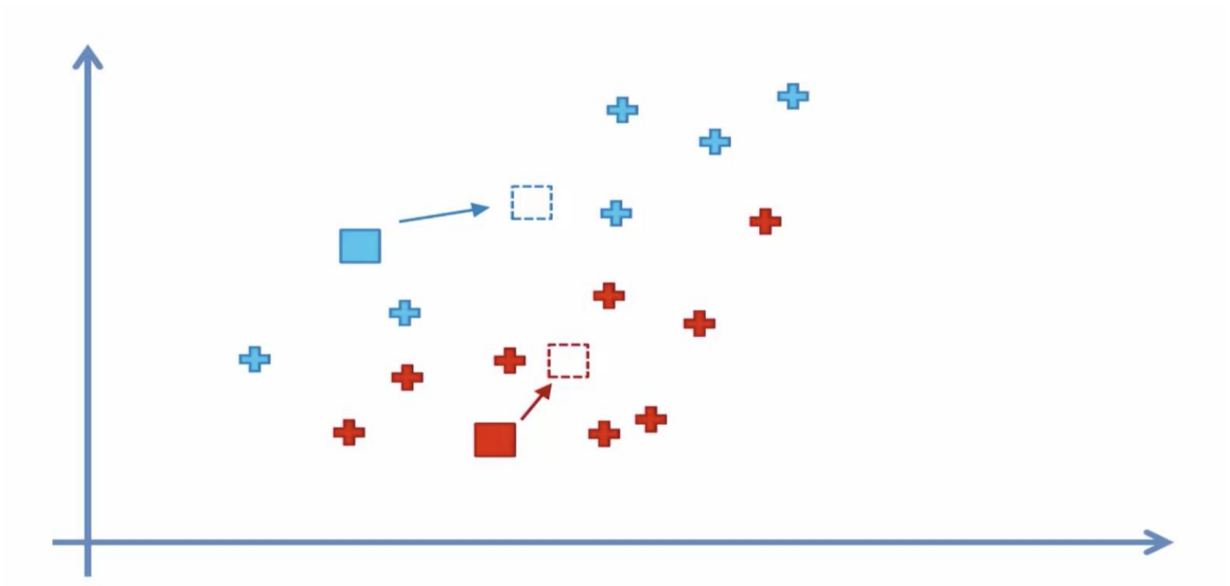
1. Select **K** (i.e. 2) random points as cluster centers called centroids



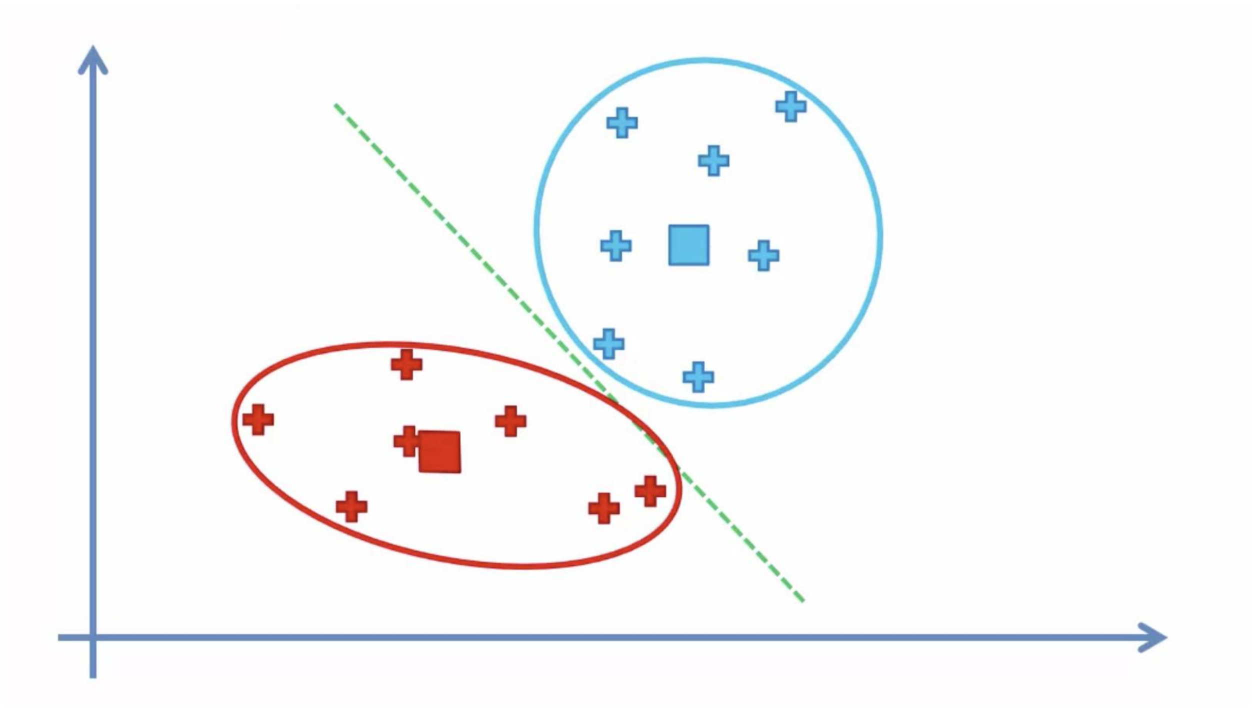
2. Assign each data point to the closest cluster by calculating its distance with respect to each centroid



3. Determine the new cluster center by computing the average of the assigned points

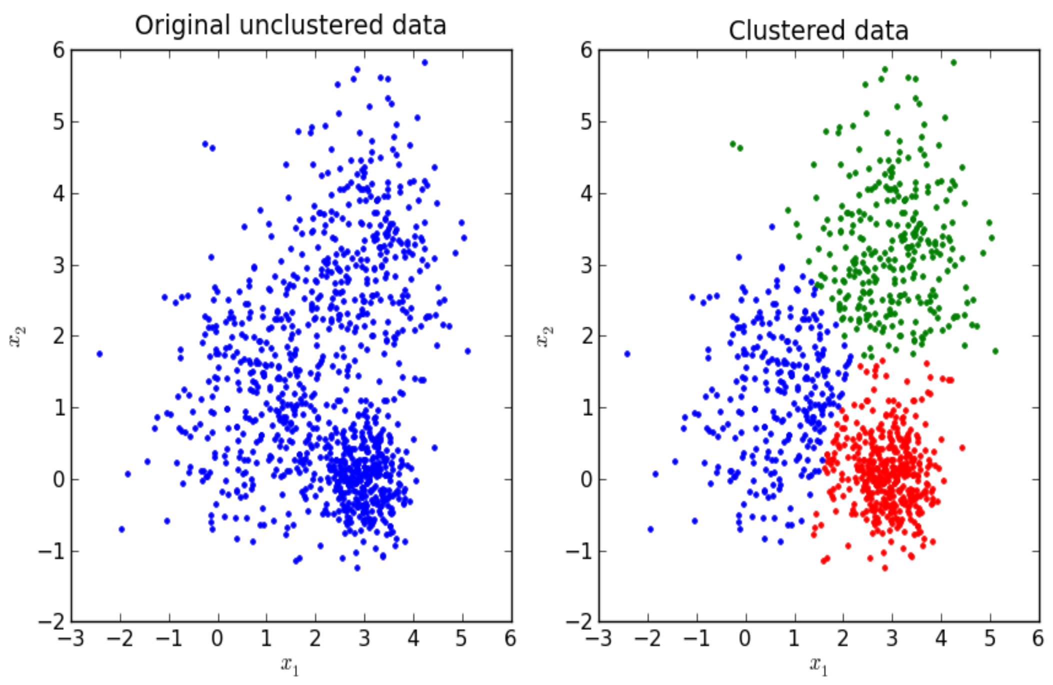


4. Repeat steps 2 and 3 until none of the cluster assignments change



K-Means falls under the category of centroid-based clustering. A centroid is a data point (imaginary or real) at the center of a cluster. In centroid-based clustering, clusters are represented by a central vector or a centroid. This centroid might not necessarily be a member of the dataset. Centroid-based clustering is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster.

K-Means is a very simple algorithm which clusters the data into *K* number of clusters.



Why Clustering? Applications of Clustering

- Why Clustering?
  - To group similar objects / data points
  - To find homogenous sets of customers
  - To segment the data in similar groups

- Applications:
  - Marketing : Customer Segmentation & Profiling
  - Libraries : Book classification
  - Retail : Store Categorization

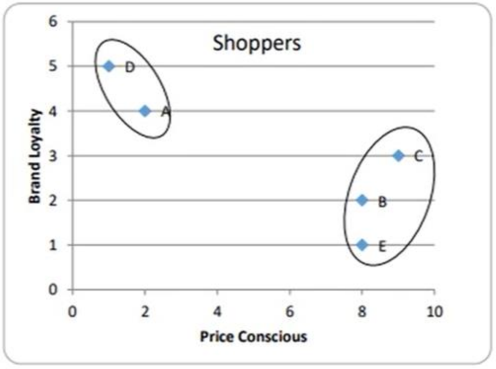
What is Clustering?

- Clustering is a technique for finding similar groups in data, called clusters.
- Clustering is an Unsupervised Learning Technique
- Clustering can also be thought of as a case reduction technique wherein it groups together similar records in cluster
- Clustering helps simplify data by reducing many data points into a few clusters (segments)

What is a Cluster?

- A *cluster* can be defined as a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters
- How do we define "Similar" in clustering?
    - Based on Distance

Shoppers	Price Conscious	Brand Loyalty
A	2	4
B	8	2
C	9	3
D	1	5
E	8	1



Distance measures

The classification of observations into groups requires some methods for computing the distance or the (dis)similarity between each pair of observations. The result of this computation is known as a dissimilarity or distance matrix. There are many methods to calculate this distance information; the choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.

The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters. The classical methods for distance measures are *Euclidean* and *Manhattan distances*

Manhattan Distance = 8 + 4 = 12

Chebyshev Distance = Max (8, 4) = 8

Eucledian Distance = sqrt (8^2 + 4^2) = 8.94

Euclidean distance:

If A $(x_1, y_1, \ldots Z_1)$ and B $(x_2, y_2, \ldots z_2)$ are cartesian coordinates

By using **Euclidean Distance** we get Distance AB as

$$D_{AB} = \sqrt{[(x_2-x_1)^2 + (y_2-y_1)^2 + \ldots + (z_2-z_1)^2]}$$

Manhattan Distance

- Manhattan Distance also called City Block Distance
- Assume two vectors: A $(x_1, y_1, \ldots z_1)$ & B $(x_2, y_2, \ldots z_2)$
- Manhattan Distance

$$= | x_2 - x_1 | + | y_2 - y_1 | + \ldots | z_2 - z_1 |$$

Chebyshev Distance

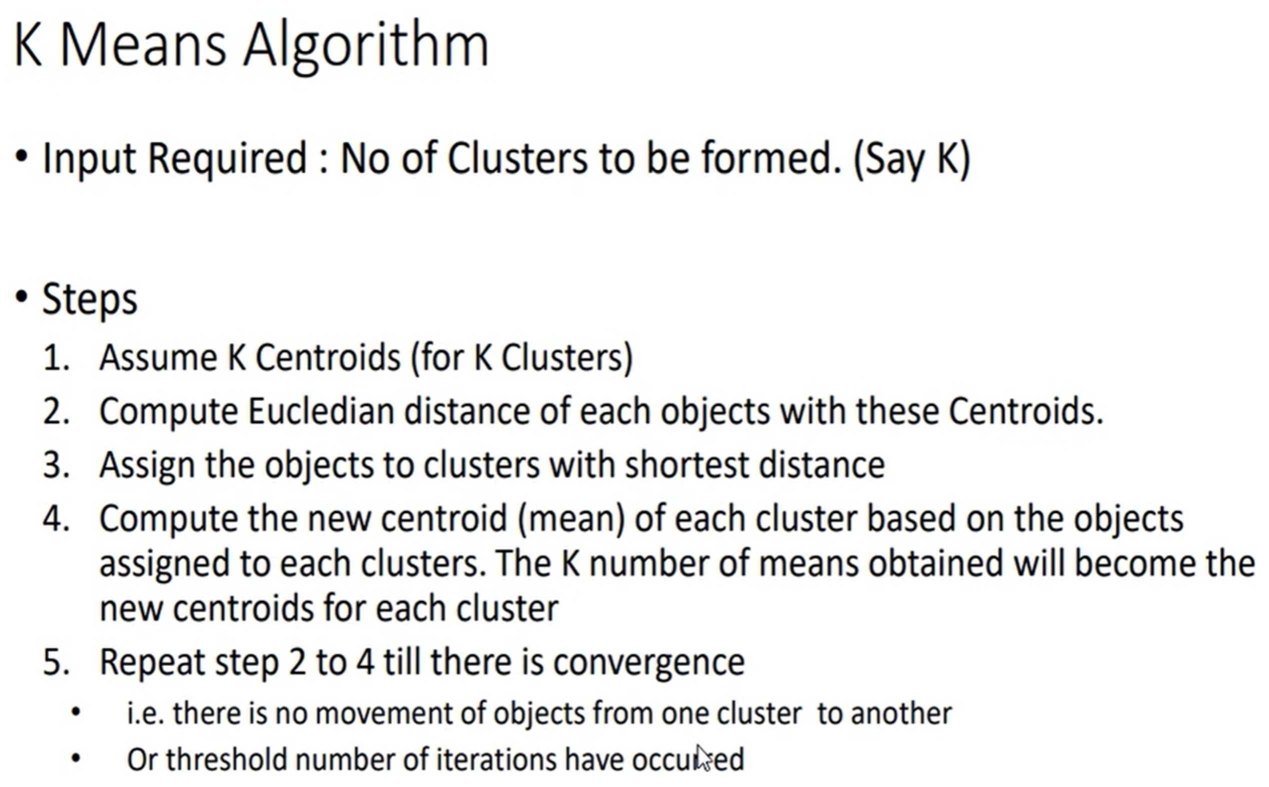
- In mathematics, **Chebyshev distance** is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension

- Assume two vectors: A $(x_1, y_1, \ldots z_1)$ & B $(x_2, y_2, \ldots z_2)$

- Chebyshev Distance

$$= \mathrm{Max}\,(| x_2 - x_1 | , | y_2 - y_1 | , \ldots | z_2 - z_1 |)$$

- Application: Survey / Research Data where the responses are Ordinal

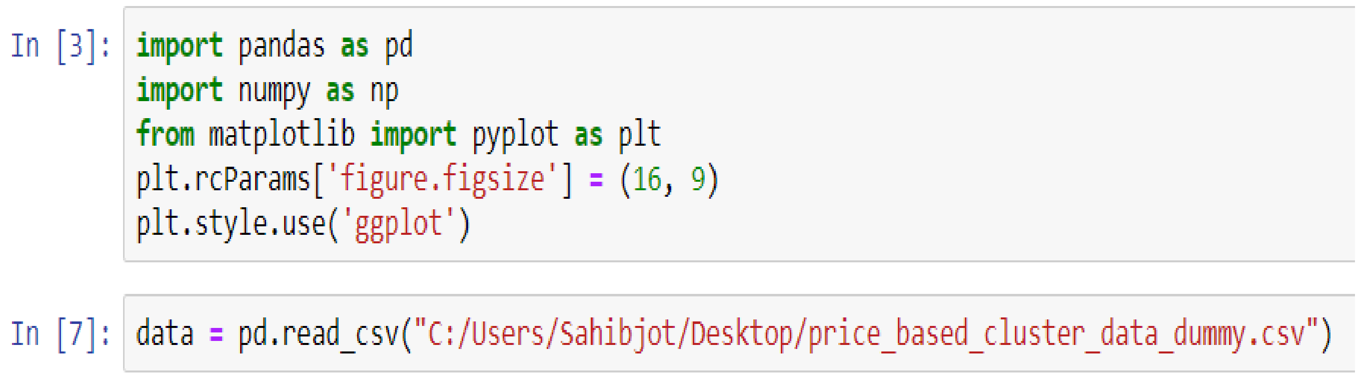


K Means Algorithm

- Input Required : No of Clusters to be formed. (Say K)

- Steps
  1. Assume K Centroids (for K Clusters)
  2. Compute Eucledian distance of each objects with these Centroids.
  3. Assign the objects to clusters with shortest distance
  4. Compute the new centroid (mean) of each cluster based on the objects assigned to each clusters. The K number of means obtained will become the new centroids for each cluster
  5. Repeat step 2 to 4 till there is convergence
     - i.e. there is no movement of objects from one cluster to another
     - Or threshold number of iterations have occured

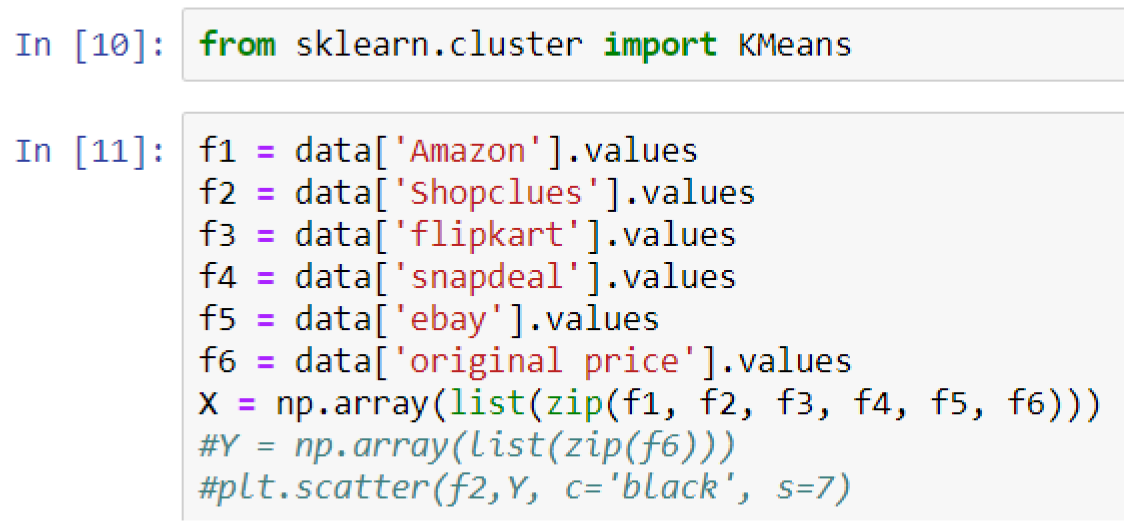
Import the libraries and the data



```
In [3]: import pandas as pd
        import numpy as np
        from matplotlib import pyplot as plt
        plt.rcParams['figure.figsize'] = (16, 9)
        plt.style.use('ggplot')
```

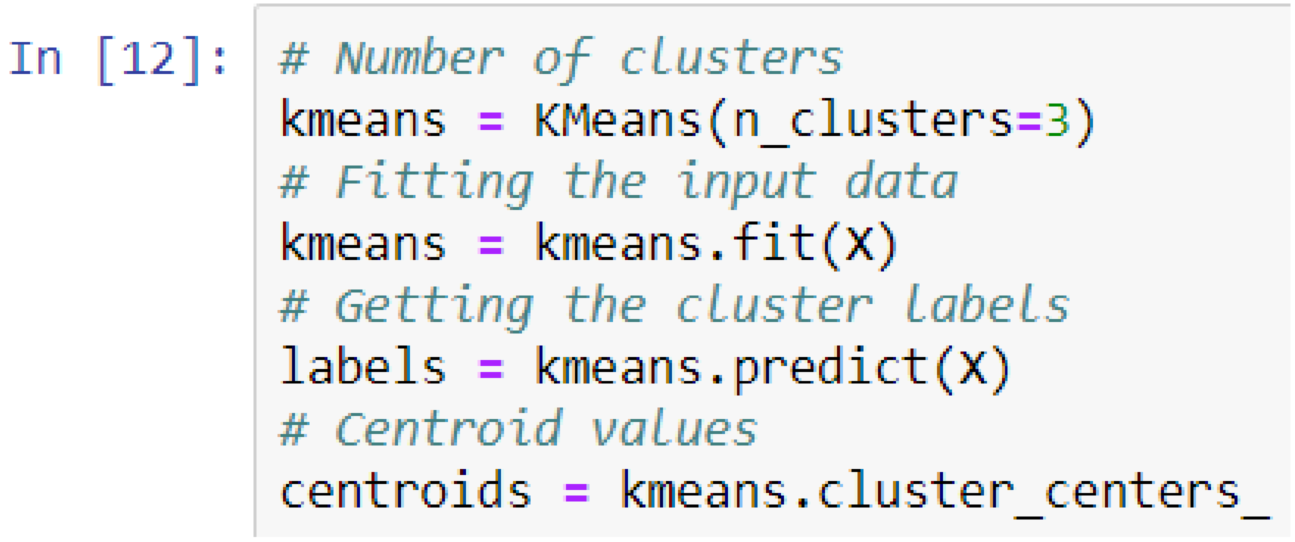
```
In [7]: data = pd.read_csv("C:/Users/Sahibjot/Desktop/price_based_cluster_data_dummy.csv")
```

Getting the values



```
In [10]: from sklearn.cluster import KMeans

In [11]: f1 = data['Amazon'].values
         f2 = data['Shopclues'].values
         f3 = data['flipkart'].values
         f4 = data['snapdeal'].values
         f5 = data['ebay'].values
         f6 = data['original price'].values
         X = np.array(list(zip(f1, f2, f3, f4, f5, f6)))
         #Y = np.array(list(zip(f6)))
         #plt.scatter(f2,Y, c='black', s=7)
```



```
In [12]: # Number of clusters
         kmeans = KMeans(n_clusters=3)
         # Fitting the input data
         kmeans = kmeans.fit(X)
         # Getting the cluster labels
         labels = kmeans.predict(X)
         # Centroid values
         centroids = kmeans.cluster_centers_
```

Plotting:

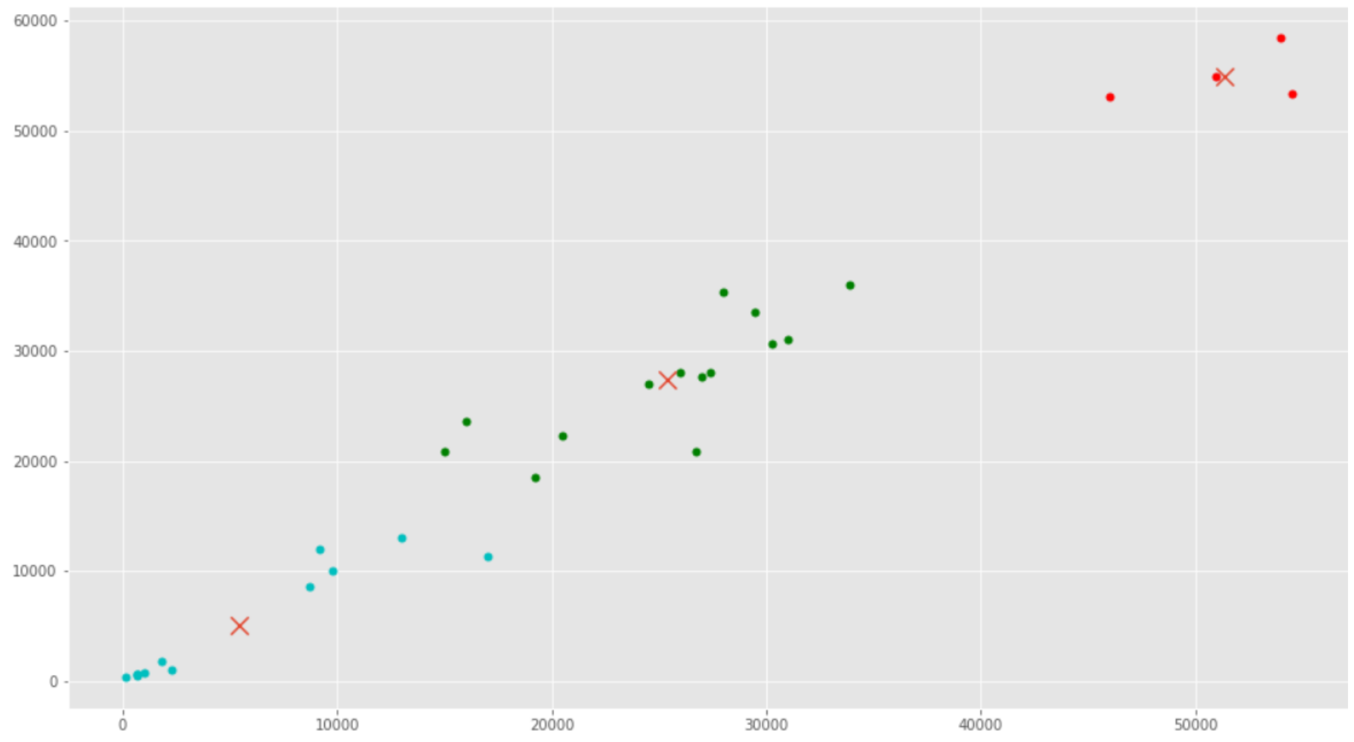
```
In [15]: colors = ["g.","r.","c.","y."]

         for i in range(len(X)):
             print("coordinate:",X[i], "label:", labels[i])
             plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize = 10)


         plt.scatter(centroids[:, 0],centroids[:, 1], marker = "x", s=150, linewidths = 5, zorder = 10)

         plt.show()
```

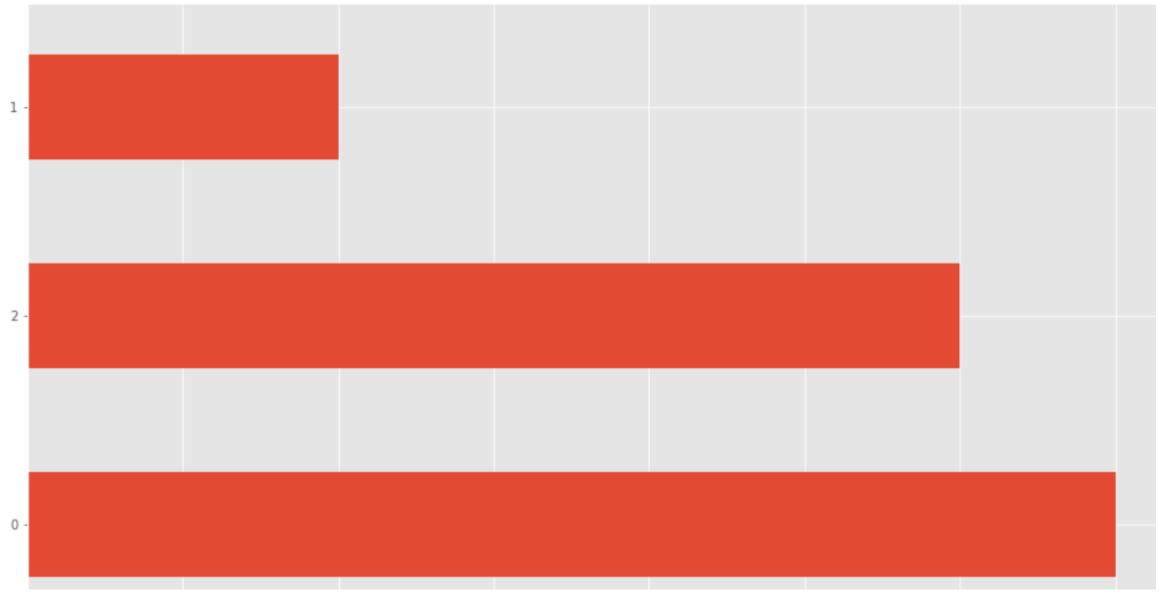
```
coordinate: [27400 27999 28999 28999 30490 35000] label: 0
coordinate: [24499 26999 18899 22199 18000 23999] label: 0
coordinate: [54490 53390 54490 61500 57500 64999] label: 1
coordinate: [26700 20899 27800 21499 25980 28999] label: 0
coordinate: [26990 27590 29990 29999 28400 35999] label: 0
coordinate: [25990 27999 22990 29000 24999 31999] label: 0
coordinate: [ 9200 11999  9790  9900 10500 12999] label: 2
coordinate: [12999 12999 12999 12900 12900 13999] label: 2
coordinate: [19196 18490 18999 19000 17999 24999] label: 0
coordinate: [20480 22288 22965 21898 22255 25999] label: 0
coordinate: [53999 58500 53900 34212 58499 61599] label: 1
coordinate: [29490 33499 29990 31499 48998 35999] label: 0
```



There are 3 clusters formed as you can see in the above graph

```
In [16]: cluster=KMeans(n_clusters=3)
         data['cluster']=cluster.fit_predict(data[['Amazon', 'Shopclues', 'flipkart', 'snapdeal', 'ebay', 'original price']])
         data.cluster.value_counts().plot(kind='barh')

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1daf4457828>
```



Standard scaling:

The StandardScaler assumes your data is normally distributed within each feature and will scale them such that the distribution is now centered around 0, with a standard deviation of 1.

```
In [19]: from sklearn.preprocessing import StandardScaler

In [20]: sc=StandardScaler()

In [21]: fonder=sc.fit_transform(data[['Amazon', 'Shopclues', 'flipkart', 'snapdeal', 'ebay', 'original price']])
```

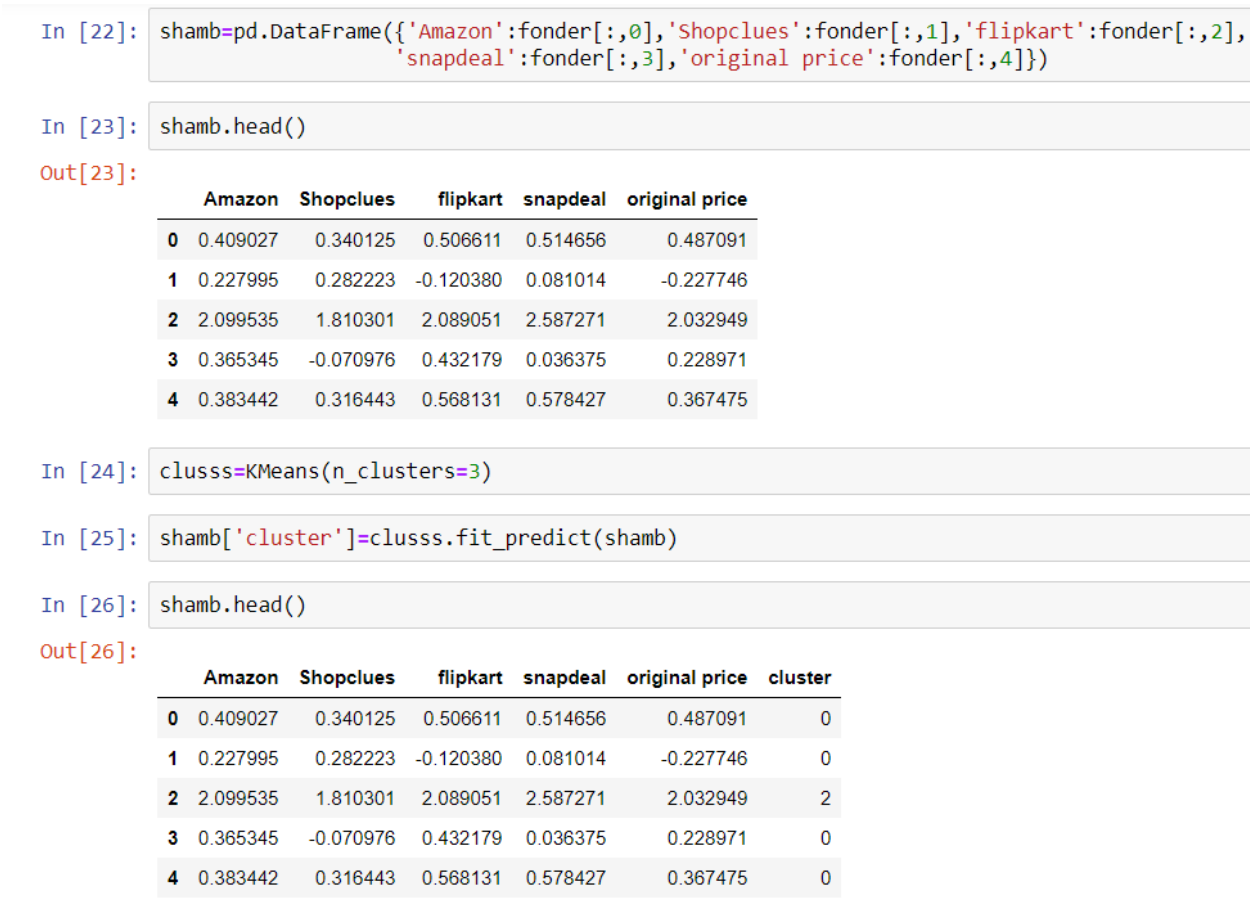
Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach k-means follows to solve the problem is called **Expectation-Maximization**. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.



```
In [22]: shamb=pd.DataFrame({'Amazon':fonder[:,0],'Shopclues':fonder[:,1],'flipkart':fonder[:,2],
                              'snapdeal':fonder[:,3],'original price':fonder[:,4]})

In [23]: shamb.head()

Out[23]:
```

	Amazon	Shopclues	flipkart	snapdeal	original price
0	0.409027	0.340125	0.506611	0.514656	0.487091
1	0.227995	0.282223	-0.120380	0.081014	-0.227746
2	2.099535	1.810301	2.089051	2.587271	2.032949
3	0.365345	-0.070976	0.432179	0.036375	0.228971
4	0.383442	0.316443	0.568131	0.578427	0.367475

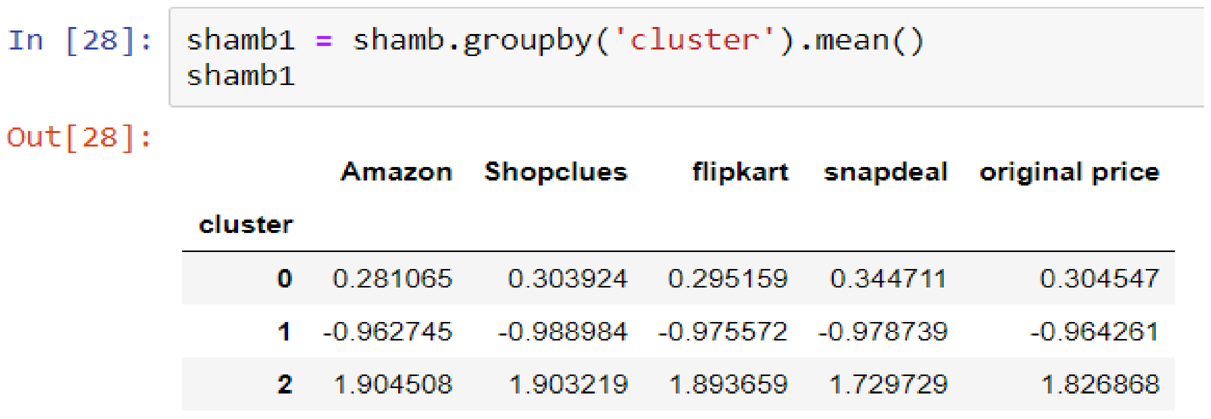
```
In [24]: clusss=KMeans(n_clusters=3)

In [25]: shamb['cluster']=clusss.fit_predict(shamb)

In [26]: shamb.head()

Out[26]:
```

	Amazon	Shopclues	flipkart	snapdeal	original price	cluster
0	0.409027	0.340125	0.506611	0.514656	0.487091	0
1	0.227995	0.282223	-0.120380	0.081014	-0.227746	0
2	2.099535	1.810301	2.089051	2.587271	2.032949	2
3	0.365345	-0.070976	0.432179	0.036375	0.228971	0
4	0.383442	0.316443	0.568131	0.578427	0.367475	0



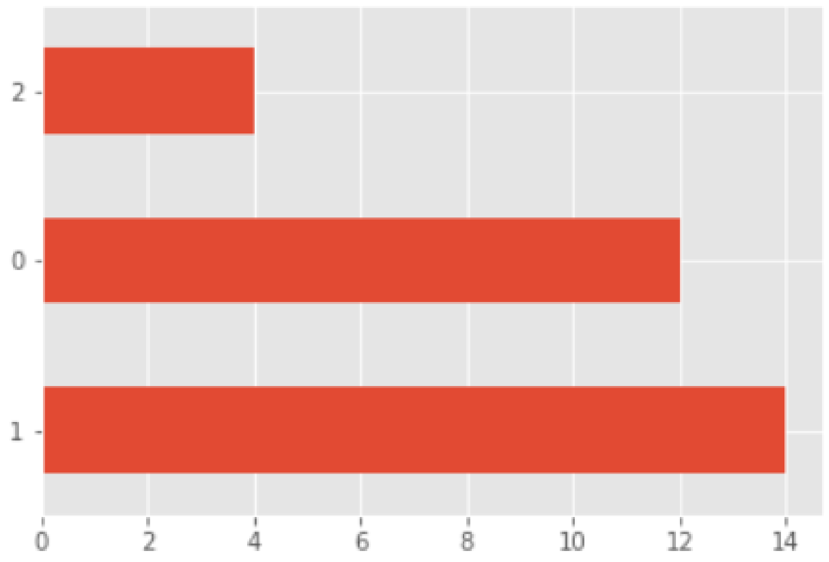
```
In [28]: shamb1 = shamb.groupby('cluster').mean()
         shamb1

Out[28]:
```

	Amazon	Shopclues	flipkart	snapdeal	original price
cluster					
0	0.281065	0.303924	0.295159	0.344711	0.304547
1	-0.962745	-0.988984	-0.975572	-0.978739	-0.964261
2	1.904508	1.903219	1.893659	1.729729	1.826868

```
In [22]: shamb.cluster.value_counts().plot(kind='barh')
         #grid()

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x2ad4a687ba8>
```


2
0
1
0
2
4
6
8
10
12
14