

Practical No. 7

Title: Implementation of K-Means clustering algorithm.

Aim: Understanding basics of K-Means clustering algorithm.

Introduction:

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

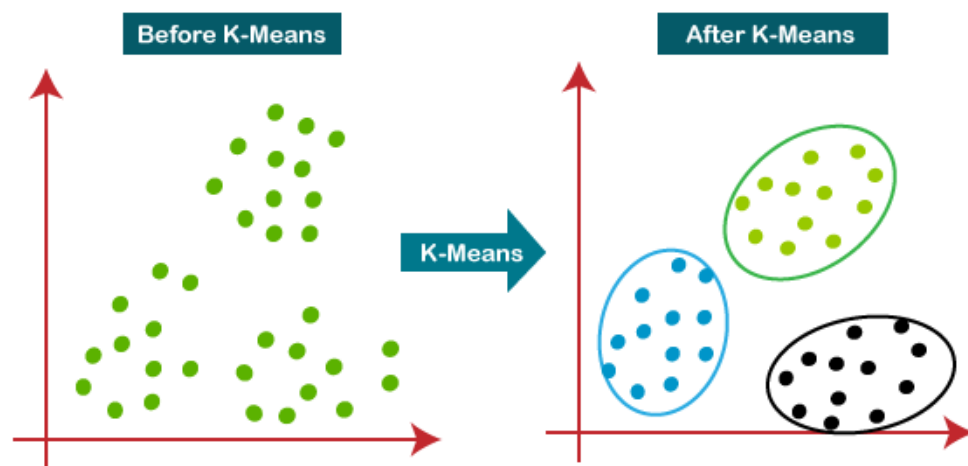
The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.

- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

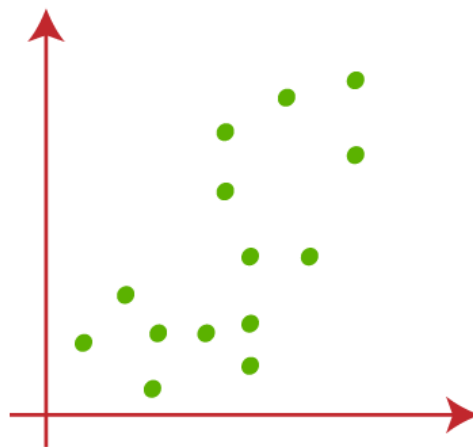
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

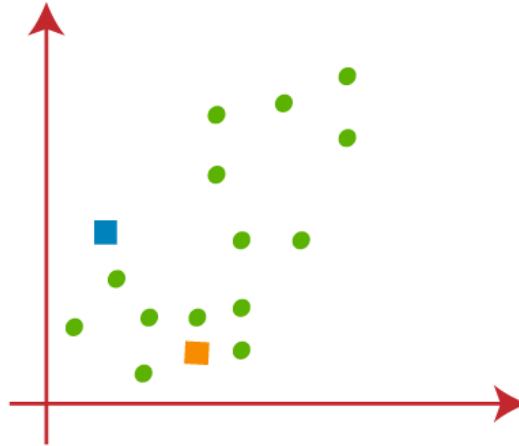
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

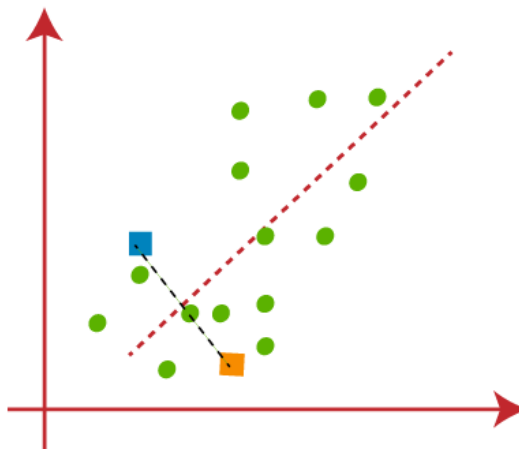


- Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset.

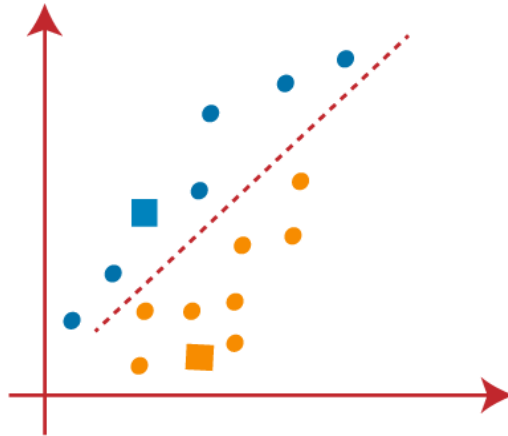
Consider the below image:



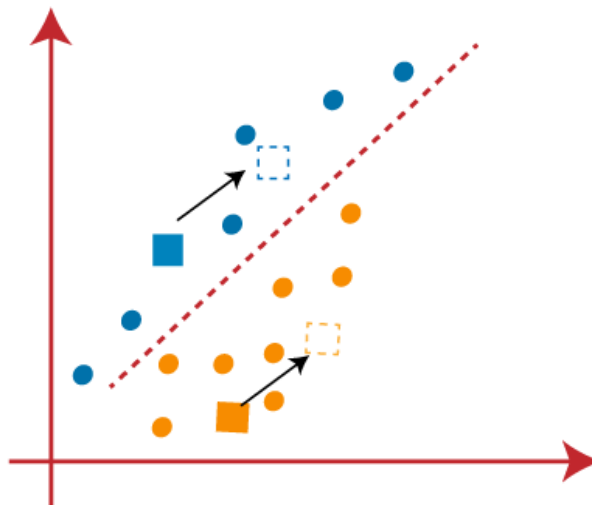
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



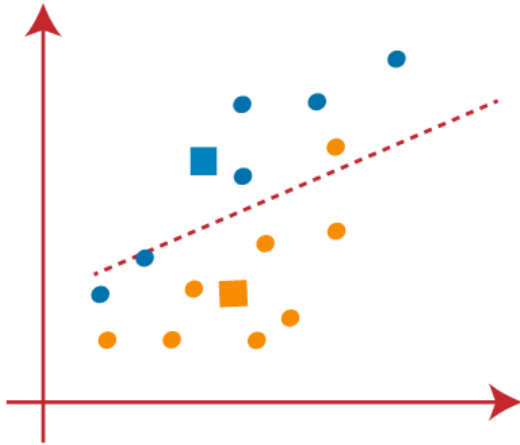
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



- As we need to find the closest cluster, so we will repeat the process by choosing a **new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



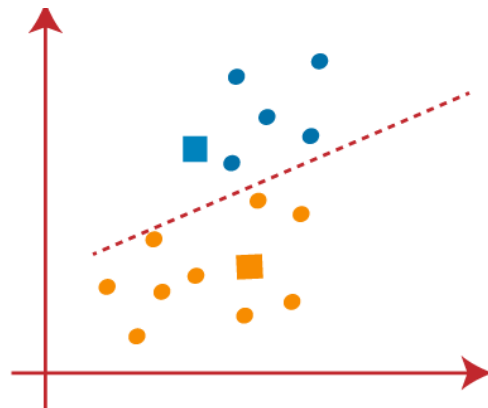
- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

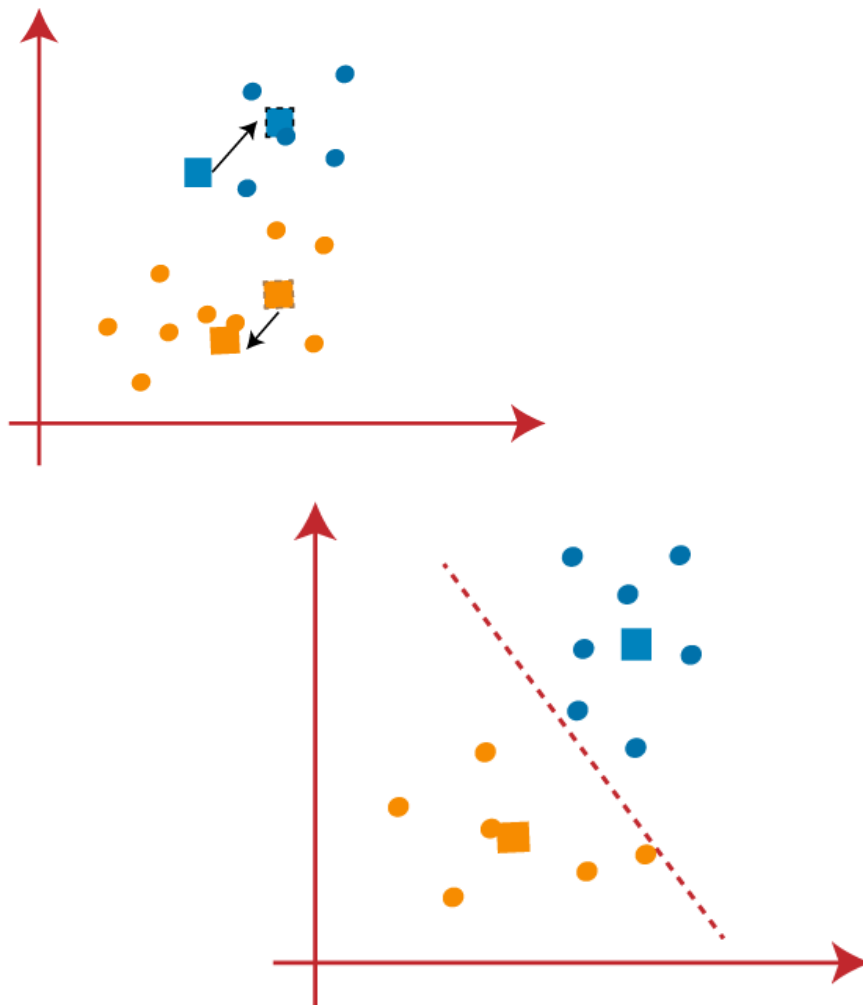
- We will repeat the process by finding the center of gravity of centroids, so the new



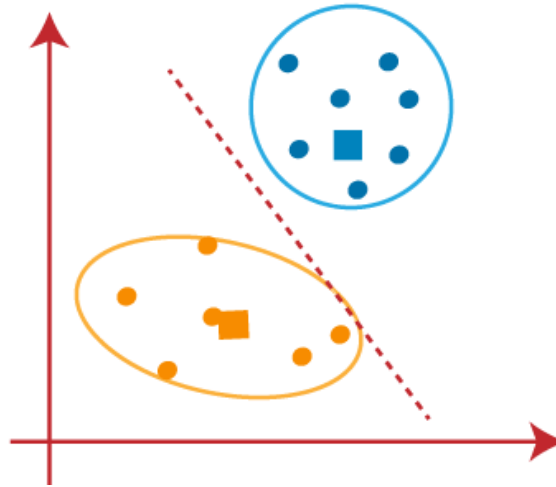
centroids will be as shown in the below image:

As we got the new centroids so again will draw the median line and reassign the data points.

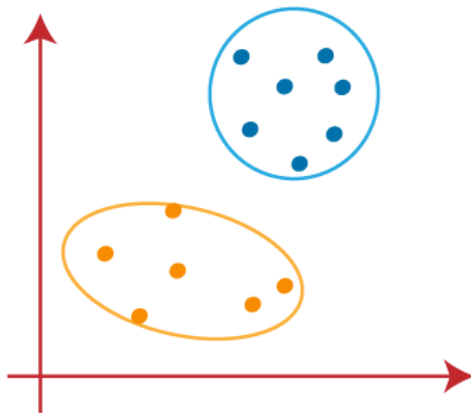
So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster.

Exercise -

Implement Clustering With K Means

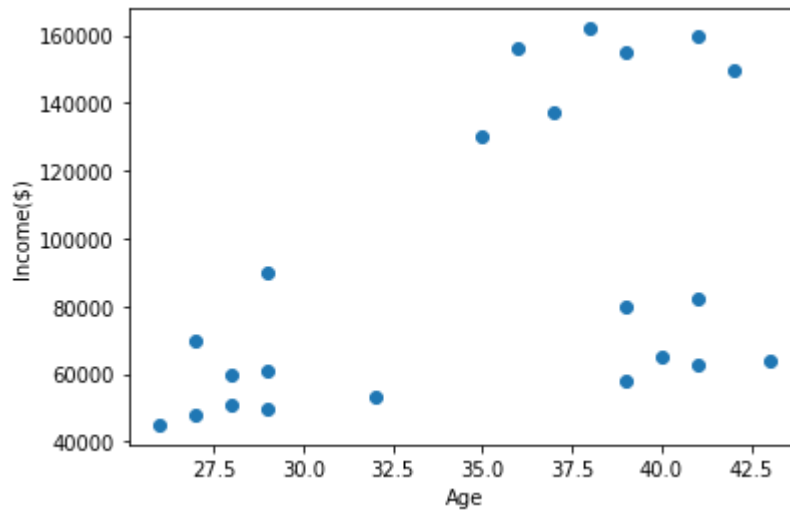
Implementation & Output:**Program:**

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt

url='https://raw.githubusercontent.com/codebasics/py/master/ML/13_kmeans/income.csv'

df = pd.read_csv(url)
df.head()

plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
```



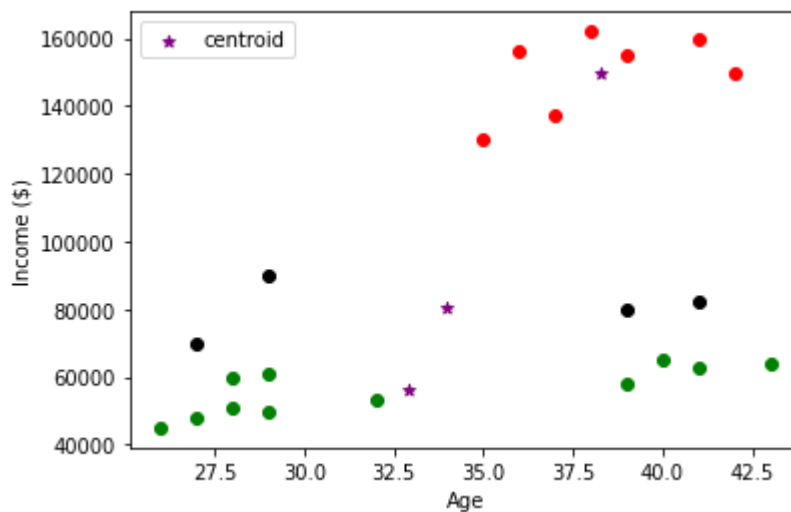
```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
```

```
df['cluster']=y_predicted
df.head()
```

```
km.cluster_centers_
```

```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
```

```
plt.scatter(km.cluster_centers_[0],km.cluster_centers_[1],color='purple',marker='*',label='centroid')  
  
plt.xlabel('Age')  
plt.ylabel('Income ($)')  
plt.legend()
```



```
scaler = MinMaxScaler()
```

```
scaler.fit(df[['Income($)']])
```

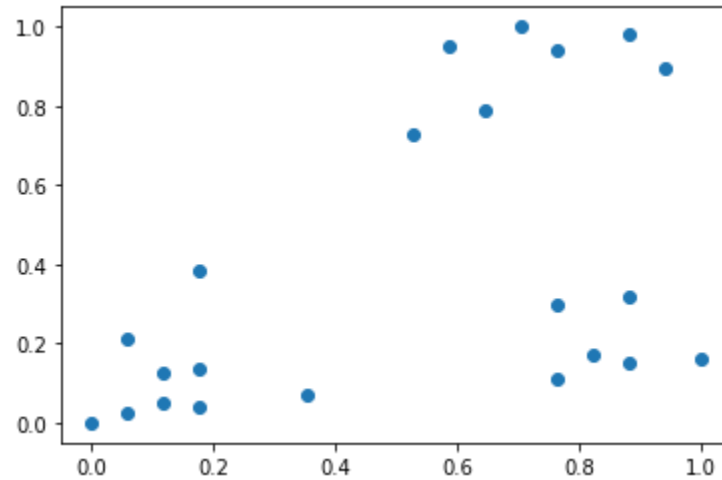
```
df['Income($)'] = scaler.transform(df[['Income($)']])
```

```
scaler.fit(df[['Age']])
```

```
df['Age'] = scaler.transform(df[['Age']])
```

```
df.head()
```

```
plt.scatter(df.Age,df['Income($)'])
```



```
km = KMeans(n_clusters=3)
```

```
y_predicted = km.fit_predict(df[['Age','Income($)']])
```

```
y_predicted
```

```
df['cluster']=y_predicted
```

```
df.head()
```

```
km.cluster_centers_
```

```
df1 = df[df.cluster==0]
```

```
df2 = df[df.cluster==1]
```

```
df3 = df[df.cluster==2]
```

```
plt.scatter(df1.Age,df1['Income($)'],color='green')
```

```
plt.scatter(df2.Age,df2['Income($)',color='red')  
plt.scatter(df3.Age,df3['Income($)',color='black')  
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label  
='centroid')  
plt.legend()
```

