K Means

- K-means clustering is an unsupervised algorithm that groups unlabelled data into different clusters. The K in its title represents the number of clusters that will be created. This is something that should be known prior to the model training.
- The algorithm is centroid-based, meaning that each data point is assigned to the cluster with the closest centroid. This algorithm can be used for any number of dimensions as we calculate the distance to centroids using the Euclidian distance.
- The benefits of the k-means algorithm are that it is easy to implement, it scales to large datasets, it will always converge, and it fits clusters with varying shapes and sizes.
- In some cases, K is not clearly defined, and we have to think about the optimal number of K. K Means clustering performs best data is well separated. When data points overlapped this clustering is not suitable.
- Some disadvantages of the model are that the number of clusters is chosen manually, the clusters are dependent on initial values, and that it is sensitive to outliers.

Objective of k-means clustering

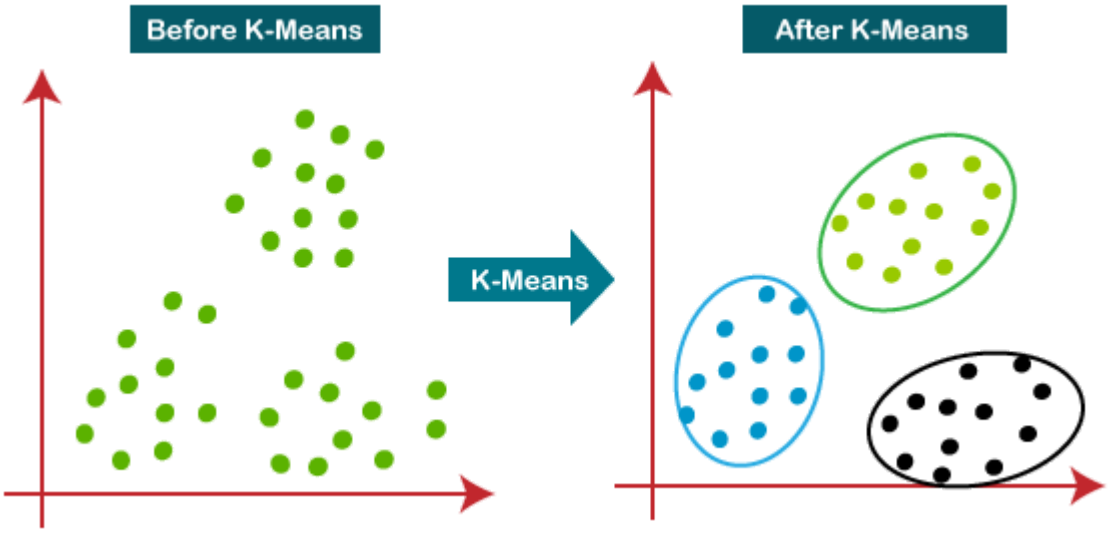
The goal of **clustering** is to divide the population or **set** of data points into a number of groups so that the data points within each group are more **comparable** to one another and different from the data points within the other groups. It is essentially a grouping of things based on how similar and different they are to one another.

Working of k-means algorithm

The algorithm works as follows:

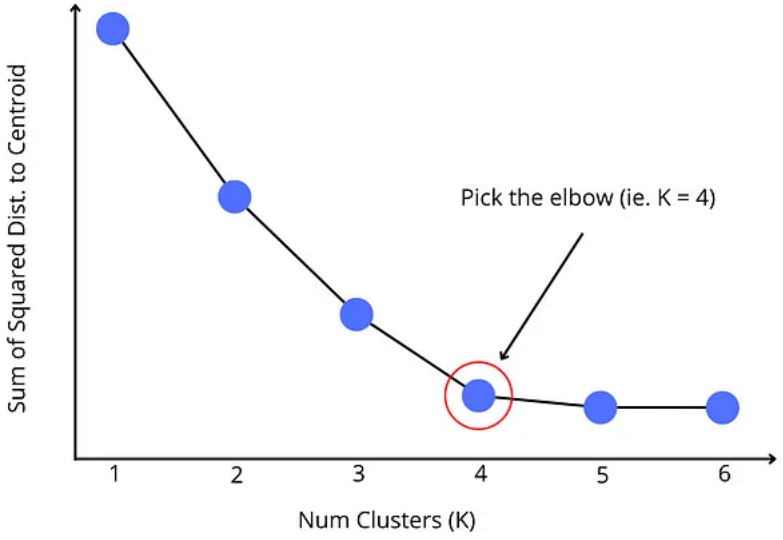
1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean, and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.

3. We repeat the process for a given number of iterations and at the end, we have our clusters.



Step 1: Determine the number of clusters (K=?)

- It is best if K is known before model training, but if not, there are strategies to find K. The most common is the elbow method, which plots the sum of the squared distances as K increases. By looking at the plot, there should be a point where increasing the size of the cluster provides minimal gain to the error function. This is known as the elbow and should be chosen as the value for K.



Step 2: Initialize cluster centroids

The next step is to initiate K centroids as the centers of each cluster. The most common initialization strategy is called Forgy Initialization. This is when the centroids for each cluster are initiated as random data points from the dataset. This converges quicker than random initialization as clusters are more likely to be present near data points.

More complex strategies like k-means++ initialization aim to choose initial points that are as far from each other as possible. This strategy has proven to be the best initialization method for the k-means algorithm.

Step 3: Assign data points to clusters

After initializing K clusters, each data point is assigned to a cluster. This is done by iterating over all the points in the data and calculating the euclidian distance to each centroid. The formula for euclidian distance works with any two points in n-dimensional euclidian space. This means that a data point with any number of dimensions is assigned to the closest cluster.

Step 4: Update cluster centroids

Once each data point is assigned to a cluster we can update the centroids of each cluster. This is done by taking the mean value of each data point in the cluster and assigning the result as the new center of the cluster.

Step 5: Iteratively Update

Then, using the newly calculated centroids we go through all the data points and re-assign them to clusters and re-calculate the centroids. This is done repeatedly until the centroid values do not change.

Python Implementation of K-means Clustering Algorithm

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

- **Importing Libraries**

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
```

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset**

Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```

- **Extracting Independent Variables**

we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
x = dataset.iloc[:, [3, 4]].values
```

Step-2: Finding the optimal number of clusters using the elbow method

we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10.

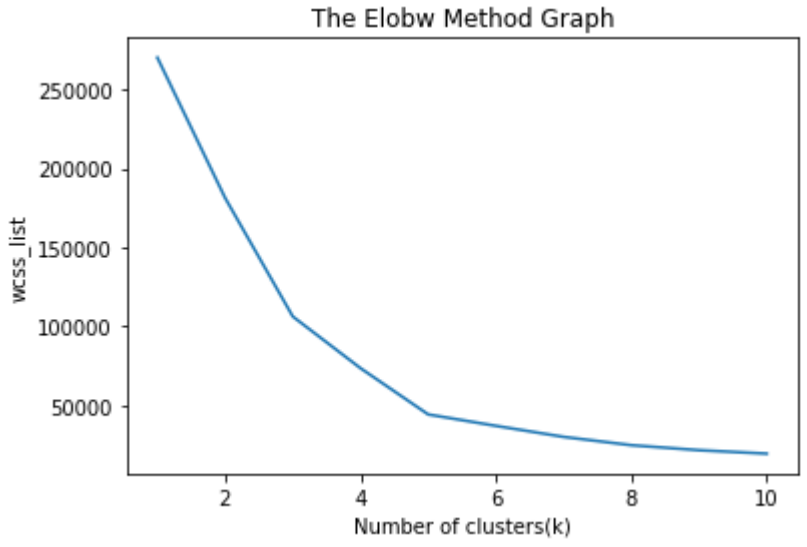
```
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS
```

```
#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_st
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10th value.



Step- 3: Training the K-means algorithm on the training dataset

To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there

are 5 clusters that need to be formed. The code is given below:

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=
y_predict= kmeans.fit_predict(x)
```

In the second line of code, we have created the dependent variable **y_predict** to train the model.

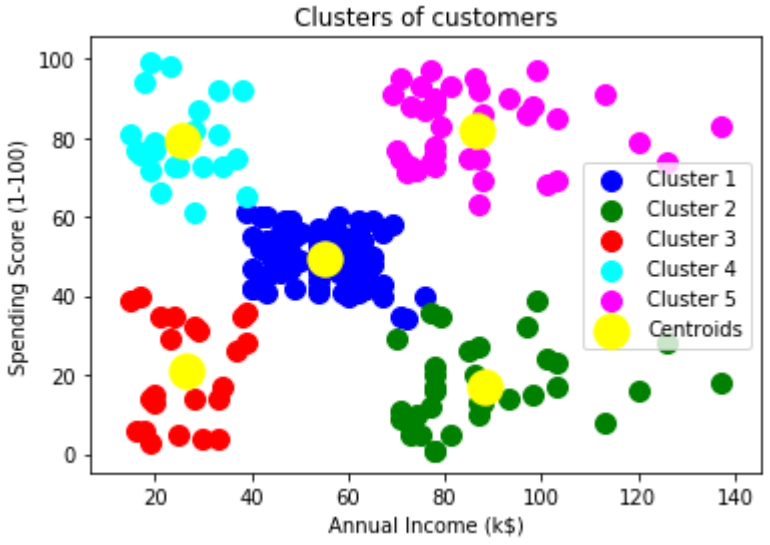
By executing the above lines of code, we will get the y_predict variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of y_predict with our original dataset.

Step-4: Visualizing the Clusters

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

OUTPUT:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target