WEEK_9_DAY_2_AFTER_NOON

Step-1: Data pre-processing Step

The first step will be the data pre-processing,

Importing Libraries

firstly, import the libraries for model, which is part of data pre-processing. The code is given below:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as plt
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, import the dataset. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers.csv')
```

The dataset looks like the below image:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
245	246	Male	30	297	69
246	247	Female	56	311	14
247	248	Male	29	313	90
248	249	Female	19	316	32
249	250	Female	31	325	86

250 rows × 5 columns

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here 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.

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

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

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

In the second step, 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. Below is the code for it:

```
#finding optimal number of clusters using the elbow method
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_state= 42)
   kmeans.fit(x)
   wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elobw Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()
```

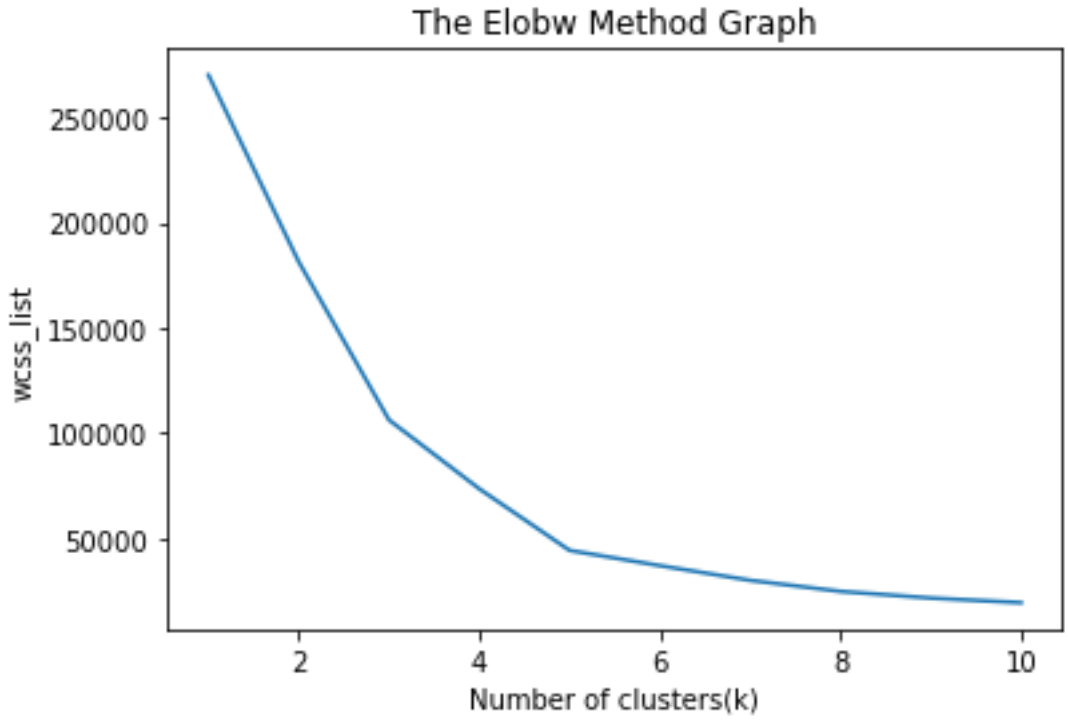
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

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 10^{th} value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**

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

As we have got the number of clusters, so we can now train the model on the dataset.

To train the model,

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

The first line is the same as above for creating the object of KMeans class.

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.

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.

```
#visulaizing the clusters
plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
plt.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
plt.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the plt.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

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
- 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**, and these customers can be the most profitable customers for the mall owner.

Evaluation Metrics

- Inertia

- Dunn Index

Inertia

The Inertia or within cluster of sum of squares value gives an indication of how coherent the different clusters are. Equation 1 shows the formula for computing the Inertia value.

$$\sum_{i=1}^{N}(x_i - C_k)^2$$

N is the number of samples within the data set, C is the center of a cluster. So the Inertia simply computes the squared distance of each sample in a cluster to its cluster center and sums them up. This process is done for each cluster and all samples within that data set. The smaller the Inertia value, the more coherent are the different clusters. When as many clusters are added as there are samples in the data set, then the Inertia value would be zero. So how to find the optimal number of clusters using the Inertia value?
For this, the so called **Elbow-Method** can be used.

Dunn Index

The Dunn index (DI) (introduced by J. C. Dunn in 1974), a metric for evaluating clustering algorithms, is an internal evaluation scheme, where the result is based on the clustered data itself. Like all other such indices, the aim of this Dunn index to identify sets of clusters that are compact, with a small variance between members of the cluster, and well separated, where the means of different clusters are sufficiently far apart, as compared to the within cluster variance.
Higher the Dunn index value, better is the clustering. The number of clusters that maximizes Dunn index is taken as the optimal number of clusters k. It also has some drawbacks. As the number of clusters and dimensionality of the data increase, the computational cost also increases.
The Dunn index for c number of clusters is defined as :

$$\text{Dunn index}(U) = \min_{1\le i\le c}\left\{\min_{1\le j\le c,\, j\ne i}\left\{\frac{\delta(X_i, X_j)}{\max_{1\le k\le c}\{\Delta(X_k)\}}\right\}\right\}$$

where,

δ (Xi, Xj) is the intercluster distance i.e. the distance between cluster Xi and Xj

Δ(Xk) is the intracluster distance of cluster Xk i.e. distance within the cluster Xk