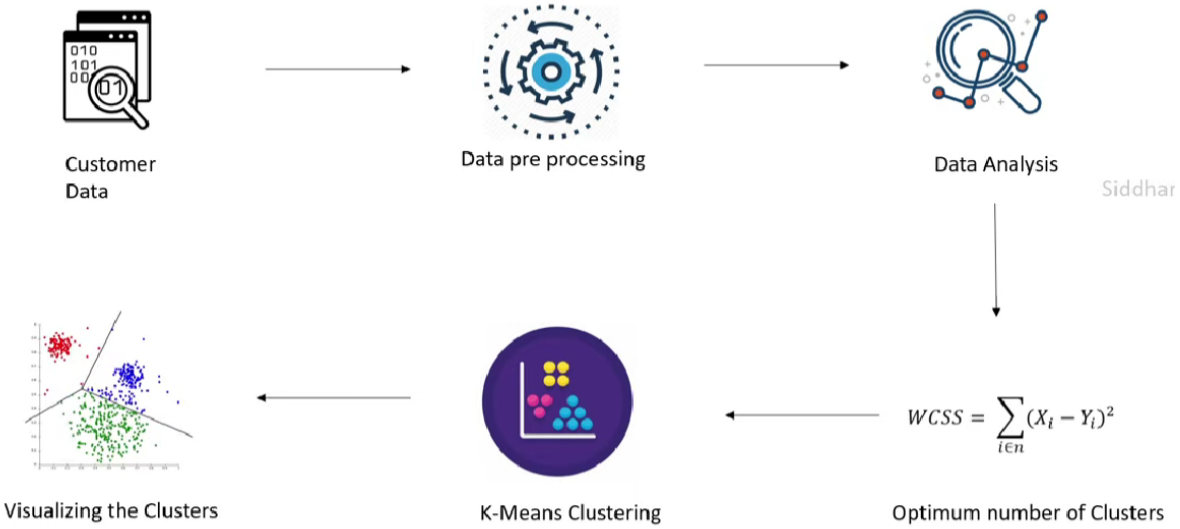
Customer Segmentation using K-Means Clustering.ipynb



STEP 1: **Importing the dependencies.**

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans
```

-NumPy can be used to perform a wide variety of mathematical operations on arrays

- Pandas is used for making data frame (A Data Frame is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet) for better processing and analysis.

-Matplotlib and seaborn are used for making plots (DATA VISUALIZATION libraries) .

-Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling.(used for importing k means algo).

STEP 2: **Data Collection & Analysis:**

-Upload your dataset

-Data set from Kaggle

(https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python) .

```
# loading the data from csv file to a Pandas DataFrame

customer_data = pd.read_csv('/content/Mall_Customers.csv')
```

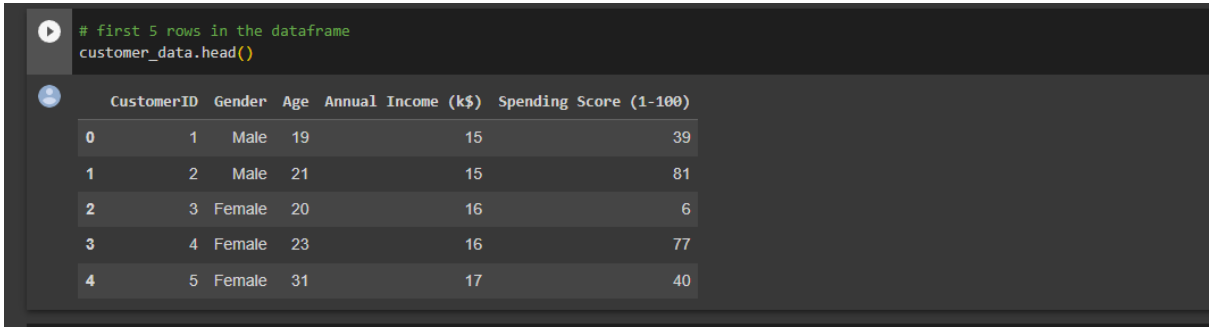
-loading the data from csv file to a Pandas DataFrame

-read_csv function will read the data

-it will load this to data frame to customers data

#first 5 rows in the dataframe

customer_data.head()



```
# first 5 rows in the dataframe
customer_data.head()
```

	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

finding the number of rows and columns

customer_data.shape

(200 , 5)

-not null means non missing values

-int64 is integer data type

-GENDER column is object and others are integer data type.

getting some informations about the dataset

customer_data.info()

-it will give no. of missing values in each column; we don't have missing values in this column but if we have any missing value then we follow methods like imputation in order to replaced those values with suitable values.

- Mode imputation replaces missing values with the mode (most frequently occurring value) of the non-missing values in the same column

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
# checking for missing values

customer_data.isnull().sum()
```

-Customer classified on basis of annual income and spending score (classification based on customer spending behaviour)

STEP 3: **Choosing the Annual Income Column & Spending Score column**

```
X = customer_data.iloc[:,[3,4]].values
```

-we are taking customer data

- we are locating particular columns which are 3 and 4(because we star from 0 which is customer id and so on).

-There are 5 columns

COLUMNS - CustomerID | Gender | Age | Annual Income | Spending Score

INDEX - (0) (1) (2) (3) (4)

- [: ,] means we are taking 3th and 4th columns , otherwise it will take 3th and 4th row.

-then print(X)

STEP 3 : Choosing the number of clusters.

WCSS -> Within Clusters Sum of Squares.

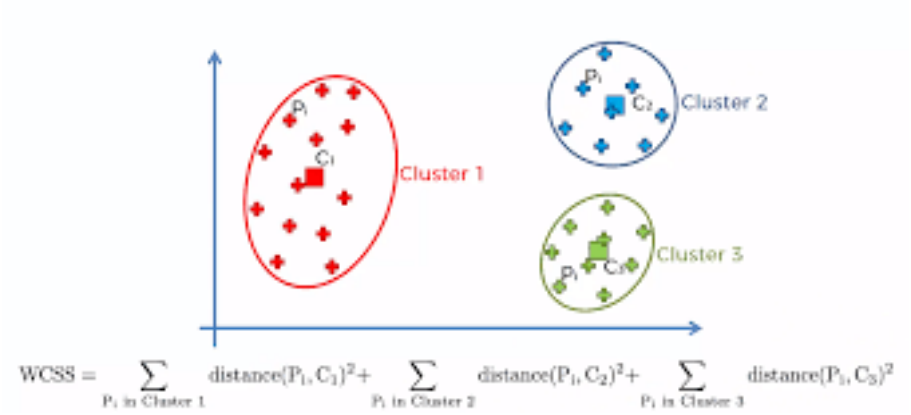
```
# finding wcss value for different number of clusters

wcss = []

for i in range(1,11):
  kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
```

kmeans.fit(X)

wcss.append(kmeans.inertia_)

- **Within Cluster Sums of Squares :** $\mathrm{WSS} = \sum_{i=1}^{N_C} \sum_{x \in C_i} d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{C_i}})^2$
- **Between Cluster Sums of Squares:** $\mathrm{BSS} = \sum_{i=1}^{N_C} |C_i| \cdot d(\bar{\mathbf{x}}_{\mathbf{C_i}}, \bar{\mathbf{x}})^2$

C_i = Cluster, N_c = # clusters, $\bar{x}_{C_i}$ = Cluster centroid, $\bar{x}$ = Sample Mean



$$\mathrm{WCSS} = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

-we are going to create a for loop for this case.

-first we find WCSS value for only one cluster and then for 2 , 3 and all the way upto 10 clusters and we will find

for which number of clusters there is minimum WCSS value.

-empty list wcss = [ ]

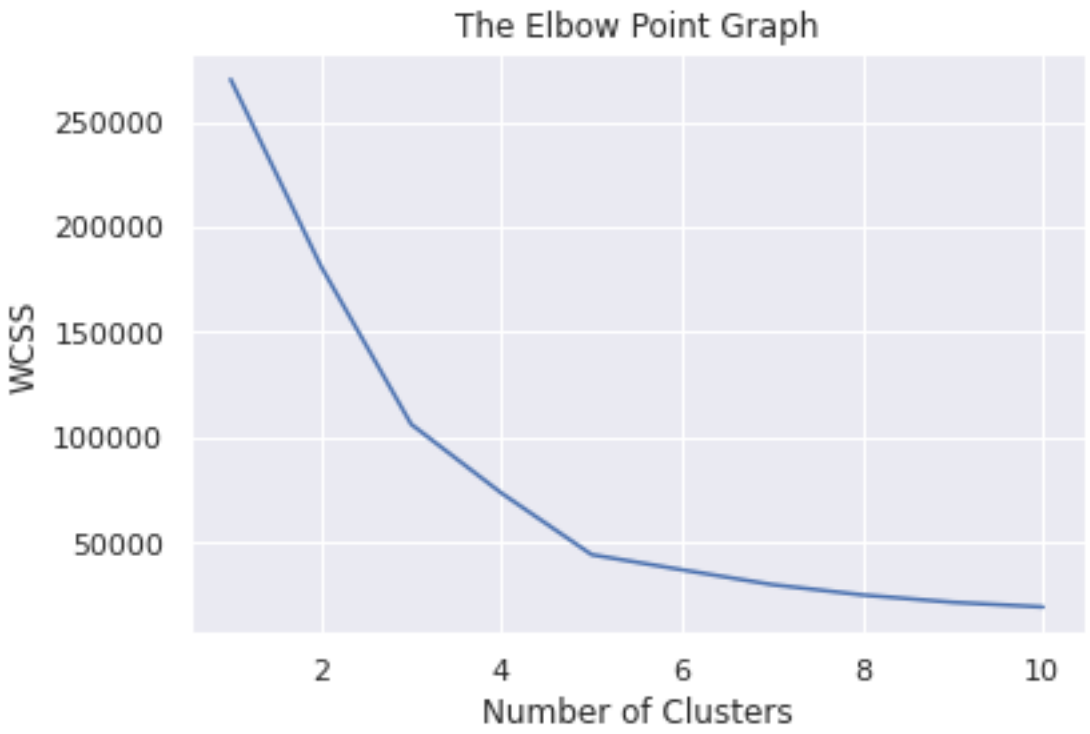
-range should be (1, 11) because it will check up to (n-1) so we want up to 10.

-init = initiation step (other initiation steps are froggy initiation, random partition etc. but this is best for this case)

-42 is a random number in random_state

- kmeans.inertia_ will give us WCSS value for each cluster.

```
# plot an elbow graph
sns.set()
plt.plot(range(1,11), wcss)
plt.title('The Elbow Point Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



- X-axis = no. of clusters
- Y-axis= WCSS
- After 5 there is no significant drop that is why we choose 5.
- Correct optimum no. of clusters is 5.

STEP 4: Optimum Number of Clusters = 5

Training the k-Means Clustering Model

```
#kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)

# return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)

print(Y)
```

-# return a label for each data point based on their cluster => we take all values of X , all these values will be splitted into 5 according to the data similarity.

- Y = kmeans.fit_(X) => we are fitting all the values of X and finding which clusters they are belong to

STEP 5: 5 Clusters - 0, 1, 2, 3, 4

Visualizing all the Clusters

-we are doing data visualization to get better understanding of this clustering project.

```
# plotting all the clusters and their Centroids

plt.figure(figsize=(8,8))
plt.scatter(X[Y==0,0], X[Y==0,1], s=50, c='green', label='Cluster 1')
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='red', label='Cluster 2')
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='yellow', label='Cluster 3')
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='violet', label='Cluster 4')
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='blue', label='Cluster 5')

# plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100, c='cyan', label='Centroids')

plt.title('Customer Groups')
plt.xlabel('Annual Income')
```

```
plt.ylabel('Spending Score')
plt.show()
```

-**Explaining How k means clustering works**

-**Steps for using the Elbow Method with K-Means**:

1.Choose K Range: Decide on a range of values for K (number of clusters).

2.Apply K-Means: Run K-Means for each K value within the chosen range.

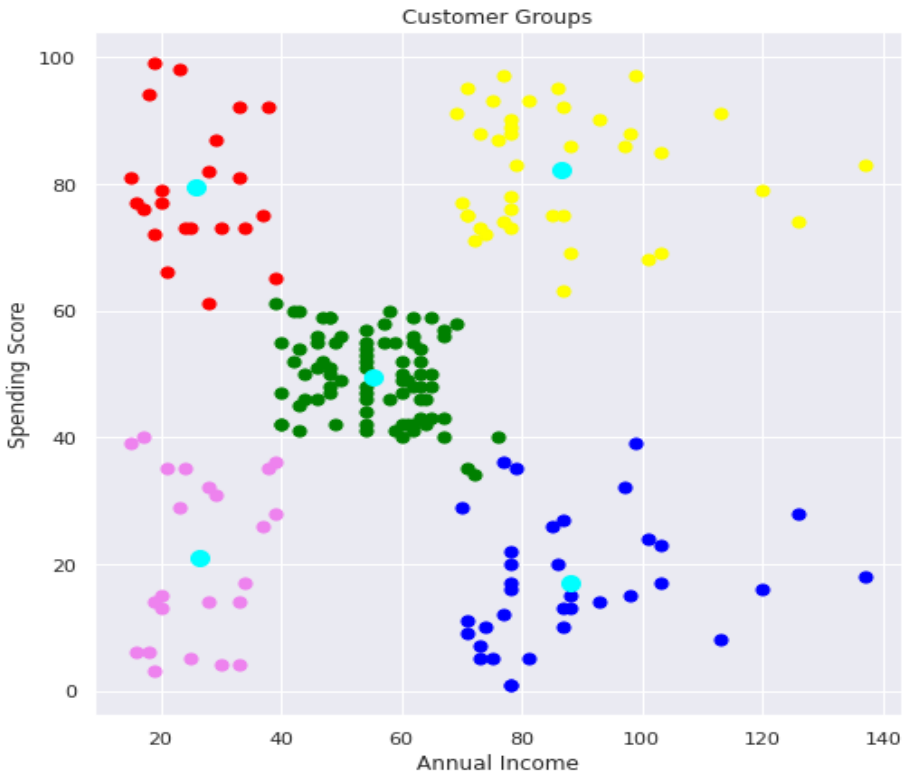
3.Calculate WCSS: Compute the within-cluster sum of squares (WCSS) for each K.

4.Plot WCSS: Create a plot of K against WCSS.

5.Find Elbow: Identify the "elbow" point where the WCSS starts to level off.

6.Select Optimal K: Choose the K value at the elbow point as the optimal number of clusters.

7.Re-run K-Means: Run K-Means with the optimal K to obtain the final clusters.



-Centroids are representation points or Mid points of each clusters.

-X axis is annual income

-Y axis Spending Score