K-Means Clustering Algorithm

INTRODUCTION

K-Means Clustering is an unsupervised learning algorithm used to solve clustering problems in machine learning and data science. The value K represents the number of predefined clusters that the algorithm aims to form. For instance, K=2 will produce two clusters, K=3 will create three clusters, and so on.

This iterative algorithm divides the dataset into KKK clusters such that each data point belongs to only one cluster based on similarity. The objective is to minimize the sum of distances between each data point and its corresponding cluster centroid.

The main tasks of the K-Means Clustering algorithm are:

1. Determine the optimal value for K (number of clusters) through iterative methods.
2. Assign each data point to the nearest cluster centroid.

WORKING

The K-Means algorithm follows these steps:

1. **Select the number K**: Decide on the number of clusters.
2. **Initialize centroids**: Choose K initial centroids randomly (these can be any points in the dataset or randomly generated).
3. **Assign clusters**: Assign each data point to the nearest centroid, forming K clusters.
4. **Update centroids**: Compute the new centroids as the mean of all data points assigned to each cluster.
5. **Repeat**: Reassign data points to the new centroids and update centroids until no data point changes clusters.
6. **Convergence**: The algorithm stops when the centroids no longer change significantly.

EXAMPLE

Suppose we have two variables M1 and M2. To visualize how K-Means works:

1. **Select K**: Choose K=2.
2. **Initialize centroids**: Randomly select two points as centroids.
3. **Assign clusters**: Assign data points to the nearest centroid and draw boundaries.
4. **Update centroids**: Calculate new centroids and reassign data points.

CHOOSING THE VALUE OF K

Determining the optimal number of clusters is crucial for effective clustering. The Elbow Method is commonly used for this purpose. It involves the following steps:

1. **Compute WCSS**: Calculate the Within-Cluster Sum of Squares (WCSS) for different values of K. WCSS measures the variance within each cluster.

   WCSS=∑(distance(Pi,Cj))^2

   Where Pi is a data point and Cj is the centroid of cluster j.

2. **Plot WCSS**: Plot WCSS values against the number of clusters K.

3. **Identify Elbow Point**: The "elbow" point on the plot indicates the optimal number of clusters.

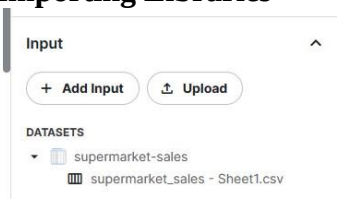
QUESTION-01

You are given a dataset containing information about customers of a supermarket, including their annual income and spending score. Your task is to perform customer segmentation using K Means clustering to identify distinct groups of customers based on their income and spending behavior.

Uploading Dataset- "supermarket_sales-Sheet1.csv" & Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

+ Code + Markdown



Display information of data

```
df=pd.read_csv("/kaggle/input/supermarket-sales/supermarket_sales - Sheet1.csv")
df
```

[2]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.3675	1/29/2019	13:46	Ewallet	40.35	4.761905	2.0175	6.2
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.4900	3/2/2019	17:16	Ewallet	973.80	4.761905	48.6900	4.4

Get overall statistics of data

```
df.head()
```

[3]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Invoice ID              1000 non-null   object
 1   Branch                  1000 non-null   object
 2   City                    1000 non-null   object
 3   Customer type           1000 non-null   object
 4   Gender                  1000 non-null   object
 5   Product line            1000 non-null   object
 6   Unit price              1000 non-null   float64
 7   Quantity                1000 non-null   int64
 8   Tax 5%                  1000 non-null   float64
 9   Total                   1000 non-null   float64
 10  Date                    1000 non-null   object
 11  Time                    1000 non-null   object
 12  Payment                 1000 non-null   object
 13  cogs                    1000 non-null   float64
 14  gross margin percentage 1000 non-null   float64
 15  gross income            1000 non-null   float64
 16  Rating                  1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
```

```
df.describe()
```

[5]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.00000
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905	15.379369	6.97270
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000	11.708825	1.71858
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905	0.508500	4.00000
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905	5.924875	5.50000
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905	12.088000	7.00000
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905	22.445250	8.50000
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905	49.650000	10.00000

Check for null values in data

```
df.isnull().sum()
```

[6]:

```
Invoice ID                0
Branch                    0
City                      0
Customer type             0
Gender                    0
Product line              0
Unit price                0
Quantity                  0
Tax 5%                    0
Total                     0
Date                      0
Time                      0
Payment                   0
cogs                      0
gross margin percentage   0
gross income              0
Rating                    0
dtype: int64
```

Removing Columns and Displaying DataFrame

```
df.drop(columns=['Invoice ID','Time'],inplace=True)
df.head()
```

[7]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Payment	cogs	gross margin percentage	gross income	Rating
0	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	Ewallet	522.83	4.761905	26.1415	9.1
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	Cash	76.40	4.761905	3.8200	9.6
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	Credit card	324.31	4.761905	16.2155	7.4
3	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	Ewallet	465.76	4.761905	23.2880	8.4
4	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	Ewallet	604.17	4.761905	30.2085	5.3

Unique Values of 'gross margin percentage' Column

```
df['gross margin percentage'].unique()
```

[8]:
```
array([4.76190476])
```

Displaying First 5 Rows of DataFrame

```
df.head(5)
```

[9]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Payment	cogs	gross margin percentage	gross income	Rating
0	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	Ewallet	522.83	4.761905	26.1415	9.1
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	Cash	76.40	4.761905	3.8200	9.6
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	Credit card	324.31	4.761905	16.2155	7.4
3	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	Ewallet	465.76	4.761905	23.2880	8.4
4	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	Ewallet	604.17	4.761905	30.2085	5.3

Display shape of data

```
df.shape
```

[10]:
```
(1000, 15)
```

Count of Unique Values in 'Gender' Column

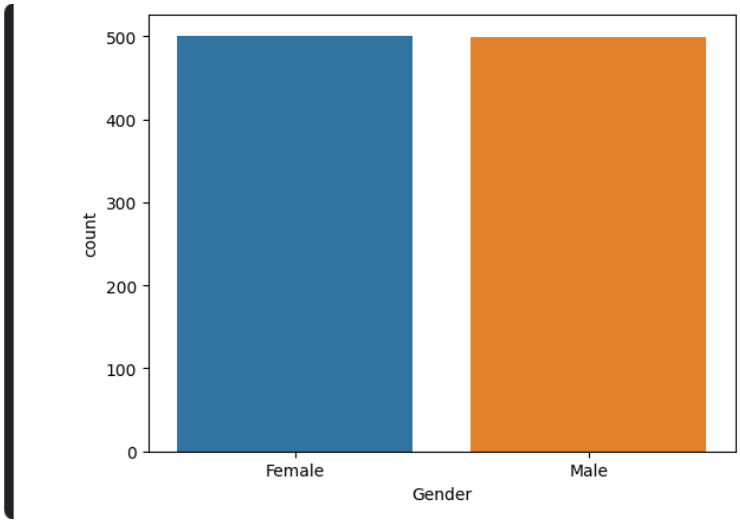
```
df['Gender'].value_counts()
```

[11]:
```
Gender
Female    501
Male      499
Name: count, dtype: int64
```

Count Plot of Gender Distribution

```
sns.countplot(x='Gender', data=df)
plt.show()
```

Output



Displaying DataFrame Columns

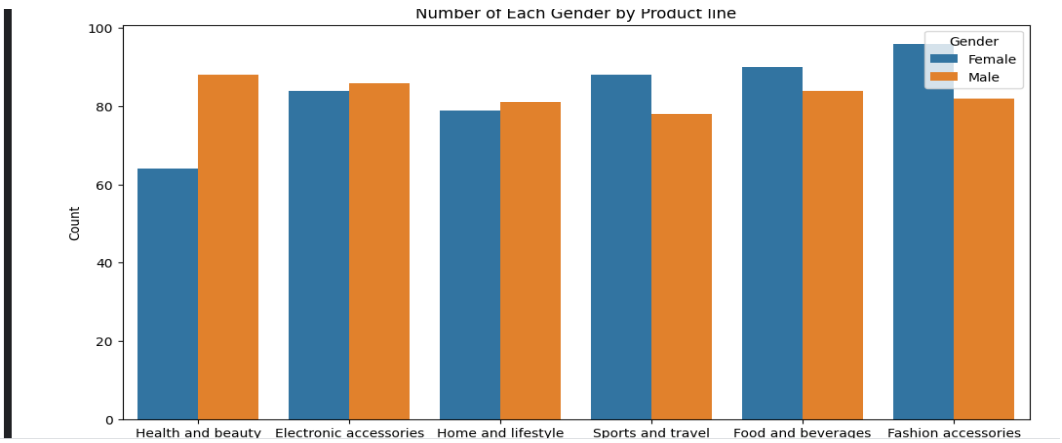
```
df.columns
```

```
[13]: Index(['Branch', 'City', 'Customer type', 'Gender', 'Product line',
       'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date', 'Payment', 'cogs',
       'gross margin percentage', 'gross income', 'Rating'],
      dtype='object')
```

Count Plot of Gender Distribution by Product Line

```
plt.figure(figsize=(12, 6))
sns.countplot(x='Product line', hue='Gender', data=df)
plt.title('Number of Each Gender by Product line')
plt.xlabel('Product line')
plt.ylabel('Count')
plt.legend(title='Gender')
plt.show()
```

Output



DataFrame of City Counts

```
place_df = pd.DataFrame(df['City'].value_counts())
place_df
```

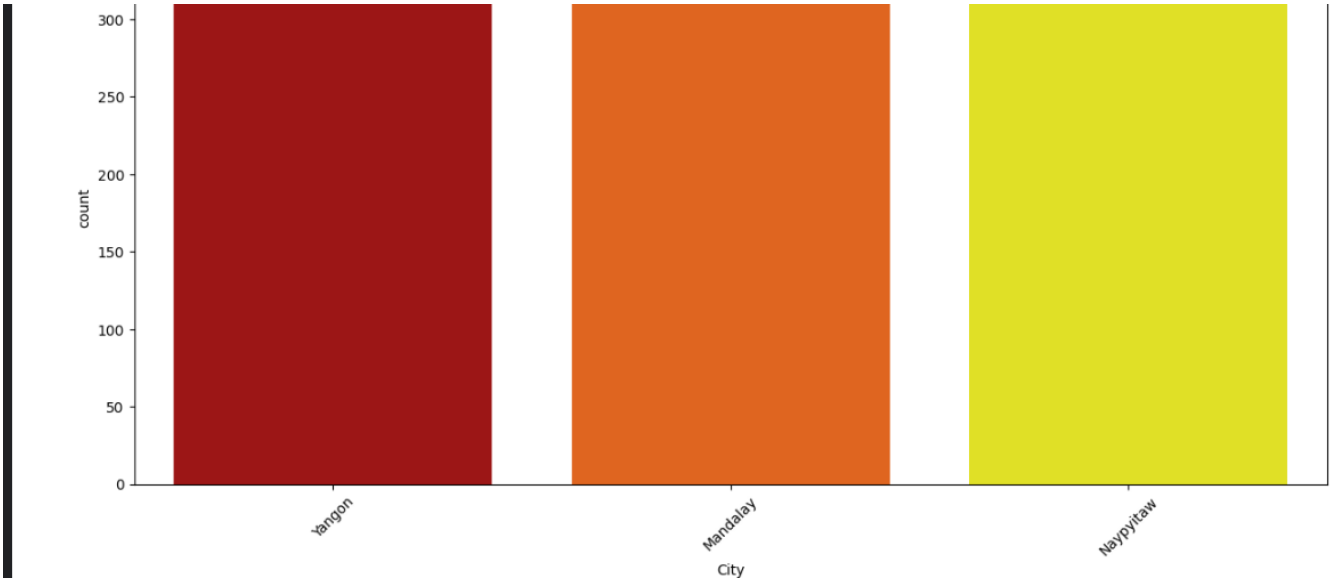
[15]:

City	count
Yangon	340
Mandalay	332
Naypyitaw	328

Bar Plot of City Counts

```
plt.figure(figsize=(15, 7))
sns.barplot(x=place_df.index, y=place_df[place_df.columns[0]], palette='hot')

plt.xlabel('City')
plt.xticks(rotation=45)
plt.show()
```

Output



DataFrame of Payment Counts

```
Payment_df = pd.DataFrame(df['Payment'].value_counts())
Payment_df
```

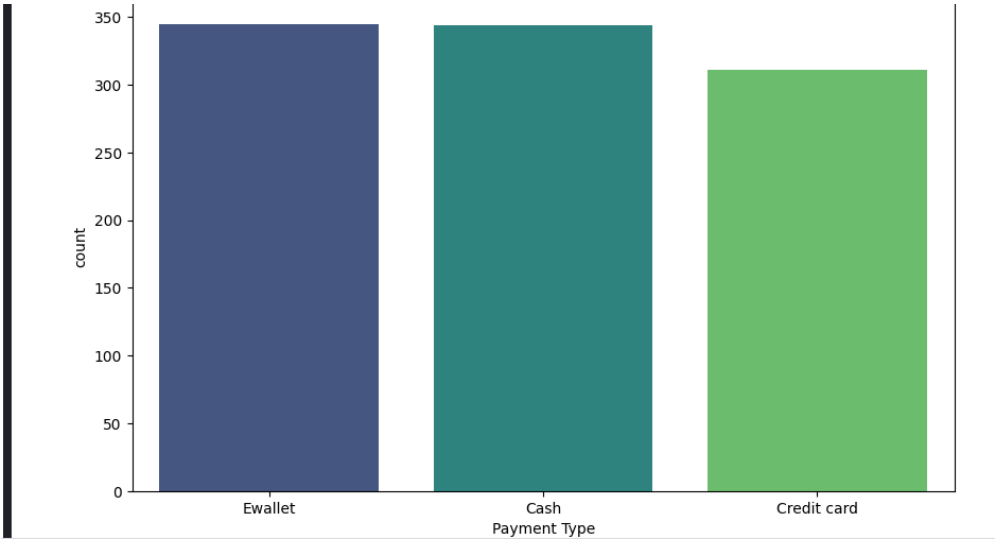
[17]:

Payment	count
Ewallet	345
Cash	344
Credit card	311

Bar Plot of Payment Types

```
plt.figure(figsize=(10, 6))
sns.barplot(x=Payment_df.index, y=Payment_df['count'], palette='viridis')

plt.xlabel('Payment Type')


plt.show()
```

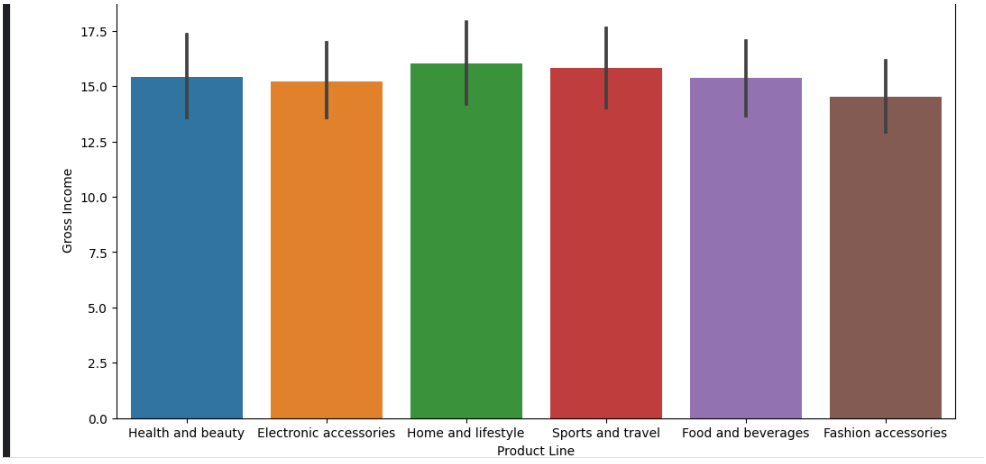
Output



Bar Plot of Gross Income by Product Line

```
plt.figure(figsize=(12, 6))
sns.barplot(x=df['Product line'], y=df['gross income'])
plt.title('Gross Income by Product Line')
plt.xlabel('Product Line')
plt.ylabel('Gross Income')

plt.show()
```

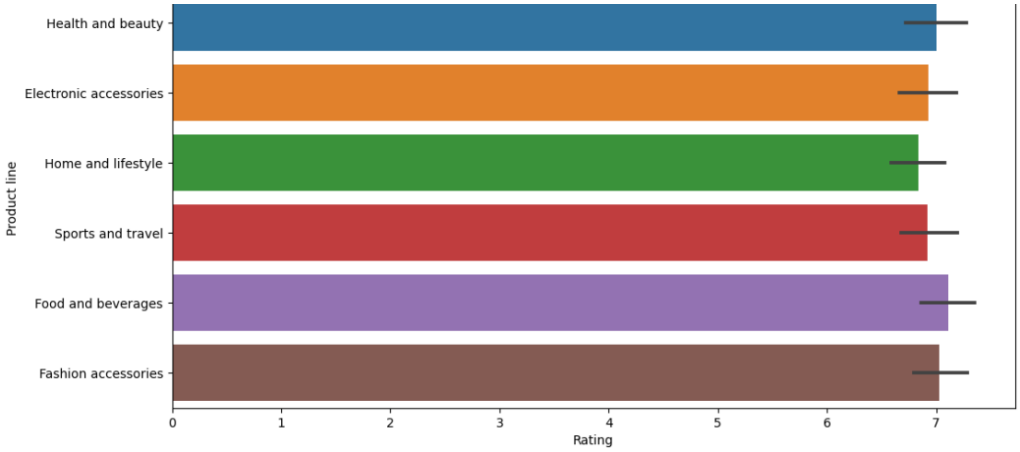
Output



Bar Plot of Ratings by Product Line

```
xdata = [0,1,2,3,4,5,6,7,8,9,10]
plt.figure(figsize = (12,6))
sns.barplot(y=df['Product line'],x = df['Rating'])
plt.show()
```

Output

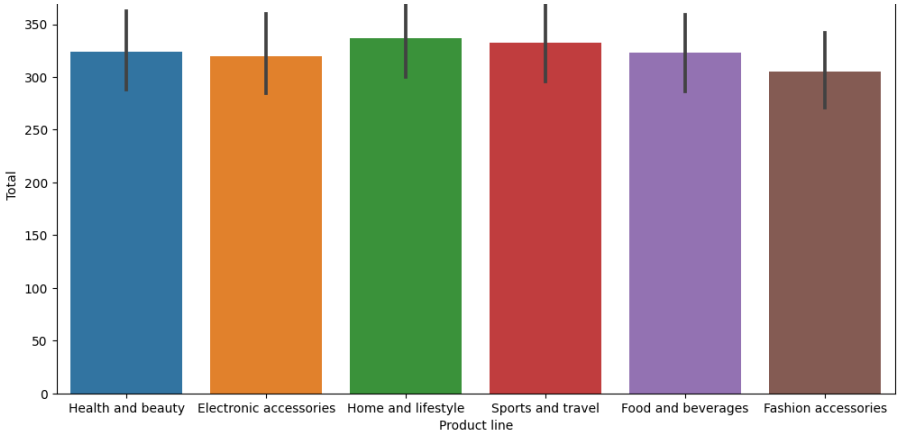


Bar Plot of Total by Product LineTop of Form

```
plt.figure(figsize = (12,6))
sns.barplot(y=df['Total'],x = df['Product line'])
```

[21]: <Axes: xlabel='Product line', ylabel='Total'>

Output

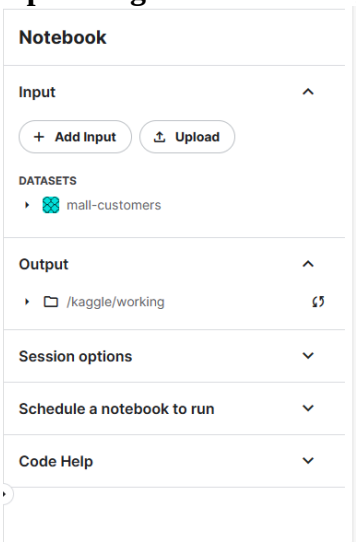


QUESTION-02

You have been provided with a dataset containing information about customers of an ecommerce platform, including their annual income and spending score. Your task is to perform customer segmentation using K-Means clustering to identify distinct groups of customers based on their income and spending behavior.

Get dataset from: **(https://www.kaggle.com/datasets/shwetabh123/mall-customers)**

Uploading Dataset- "Mall_Customers.csv"



Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data=pd.read_csv("/kaggle/input/mall-customers/Mall_Customers.csv")
data
```

[2]:

	CustomerID	Genre	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
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

Understanding the data

Display first 5 rows

```
data.head()
```

[3]:

	CustomerID	Genre	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

Display last 5 rows

```
data.tail()
```

[4]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

Display shape of data

```
data.shape
```

[5]: (200, 5)

```
print("The of rows: ", data.shape[0])
print("The of rows: ", data.shape[1])
```

```
The of rows:  200
The of rows:  5
```

Display information of data

```
data.info()
```

```
<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   Genre                   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
```

Check for null values in data

```
data.isnull().sum()
```

[8]:

```
CustomerID              0
Genre                   0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

Get overall statistics of data

```
data.describe()
```

[9]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

K-Means

```
data.columns
```

[10]:

```
Index(['CustomerID', 'Genre', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

+ Code + Markdown

[ ]:

```
# X = data[[Annual Income (k$), Spending Score (1-100)]]
```

[ ]:

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

```
X
```

[13]:

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 86,  95],
       [ 87,  27],
       [ 87,  63],
       [ 87,  13],
       [ 87,  75],
       [ 87,  10],
       [ 87,  92],
       [ 88,  13],
       [ 88,  86],
       [ 88,  15],
       [ 88,  69],
       [ 93,  14],
       [ 93,  90],
       [ 97,  32],
       [ 97,  86],
       [ 98,  15],
       [ 98,  88],
       [ 99,  39],
       [ 99,  97],
       [101,  24],
       [101,  68],
       [103,  17],
       [103,  85],
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]])
```

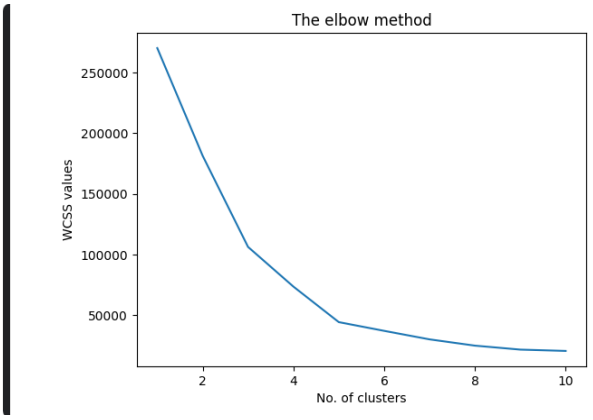
Perform elbow method to find optimal no of clusters

[16]:
```
from sklearn.cluster import KMeans
wcss=[]
```

```
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0, n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
plt.plot(range(1,11), wcss)
plt.title('The elbow method')
plt.xlabel('No. of clusters')
plt.ylabel('WCSS values')
plt.show()
```

OUTPUT



Training a model using unsupervised learning algorithm(K-Means)

[21]:
```
kmeansmodel= KMeans(n_clusters = 5, init = 'k-means++', random_state=0)
```

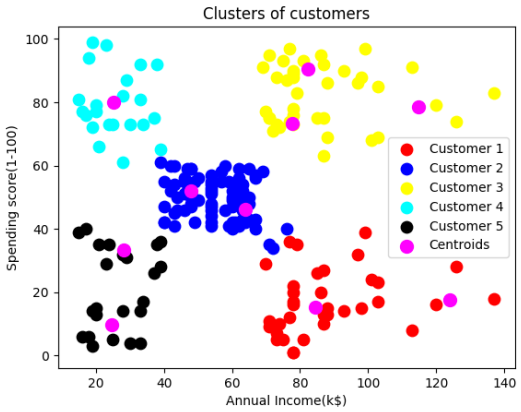
[25]:
```
import warnings

# Filter out FutureWarnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Your code here
y_means = kmeansmodel.fit_predict(X)
```

```
y_means
```

[26]:
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int32)
```

```
plt.scatter(X[y_means == 0,0], X[y_means == 0,1], s= 80, c = "red", label = 'Customer 1')
plt.scatter(X[y_means == 1,0], X[y_means == 1,1], s= 80, c = "blue", label = 'Customer 2')
plt.scatter(X[y_means == 2,0], X[y_means == 2,1], s= 80, c = "yellow", label = 'Customer 3')
plt.scatter(X[y_means == 3,0], X[y_means == 3,1], s= 80, c = "cyan", label = 'Customer 4')
plt.scatter(X[y_means == 4,0], X[y_means == 4,1], s= 80, c = "black", label = 'Customer 5')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s= 100, c= 'magenta', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income(k$)')
plt.ylabel('Spending score(1-100)')
plt.legend()
plt.show()
```

OUTPUT




```
kmeansmodel.predict([[15,39]])
```

[28]: array([4], dtype=int32)

Save the model

[30]:
```
import joblib
```

[31]:
```
joblib.dump(kmeansmodel, "customer_segmentation")
```

[31]: ['customer_segmentation']

[32]:
```
model = joblib.load("customer_segmentation")
```

```
model.predict([[20,39]])
```

[33]: array([4], dtype=int32)