Institute of Information Technology (IIT)

Jahangirnagar University

Lab Report: 08

Submitted by:

Name: Zannat Hossain Tamim

Roll No:1970

Lab Date:22.08.23

Submission Date: 28.08.23

Lab Report # Day 08

Exercise 01:

Use any dataset to use an unsupervised algorithm : K means Clustering.

- Briefly explain the k-means clustering algorithm works.
- Describe the dataset you used for clustering. What are the features? What is the source of the data?
- How did you prepare the data for the k-means algorithm? Did you normalize or standardize the features?
- Explain how you determined the optimal number of clusters k. What values did you test?
- Provide the relevant code and parameters used for implementing k-means clustering on the dataset.
- Present the cluster results visually using charts, graphs, or other methods.
- Summarize the characteristics of each cluster. What common traits define each cluster?

K-means Clustering Algorithm:

K-means clustering is an unsupervised machine learning algorithm that groups data points into a predefined number of clusters. The algorithm works by iteratively assigning data points to the cluster with the nearest mean until the assignments no longer change.

The k-means algorithm takes two inputs:

One is the number of clusters (k) and another is the distance metric.

The distance metric is used to measure the distance between data points. The most common distance metric is the Euclidean distance, but other metrics can also be used.

The k-means algorithm works as follows:

1) Initialize the cluster centroids. The algorithm randomly initializes k cluster centroids.
2) Assign data points to clusters. Each data point is assigned to the cluster with the nearest centroid.
3) Update the cluster centroids. The centroid of each cluster is updated to be the mean of the data points in that cluster.

4) Repeat steps 2 and 3 until the assignments no longer change.

The k-means algorithm is a simple and efficient algorithm that can be used to cluster data points into a predefined number of clusters. However, it has some limitations. One limitation is that the algorithm is sensitive to the initial choice of the cluster centroids. Another limitation is that the algorithm can only cluster data points that are in a spherical shape.

Here are some of the applications of k-means clustering:

1. Customer segmentation
2. Market basket analysis
3. Image segmentation
4. Text clustering
5. Gene clustering

About Dataset

The Mall Customers from Kaggle dataset is a publicly available dataset on Kaggle. It contains information about 200 customers of a mall, including their customer ID, age, gender, annual income, and spending score. Here are the variables defined in the dataset:

CustomerID: A unique integer identifier assigned to each customer in the dataset.

Genre: A categorical variable with two categories: "Male" and "Female".

Age: An integer variable that represents the age of the customer.

Annual Income (k$): A floating variable that represents the annual income of the customer in thousands of dollars.

Spending Score (1-100): An integer variable that represents the customer's spending behavior, with a score of 100 indicating the highest spending behavior.

Source of Dataset:

The data was gathered by a mall or other retail business. The information was first posted on Kaggle by a user going by the handle of Kandij.

Exploratory Data Analysis

The following are the steps on how to prepare data for the k-means algorithm:

1. Check for missing values. Missing values can skew the results of the k-means algorithm. If there are missing values, one can impute the missing values.
2. Encode categorical variables. Categorical variables need to be encoded before they can be used by the k-means algorithm. There are two common ways to encode categorical variables:

   i) Label encoding: This is the simplest way to encode categorical variables. Each category is assigned a unique integer value.

   ii) One-hot encoding: This is a more complex way to encode categorical variables. Each category is represented by a binary variable.

3. Normalize the data. The k-means algorithm works best when the data is normalized. This means that the data should be scaled to have a mean of 0 and a standard deviation of 1.
4. Choose the number of clusters (k). The number of clusters is a hyperparameter of the k-means algorithm. The optimal number of clusters depends on the data and the problem one can solve. There are a few methods to choose the number of clusters, such as the elbow method and the silhouette coefficient.

Data preparation:

Yes, I would standardize the feature.
I would have subtracted the mean and divided it by the standard deviation of each feature.

Here, the StandardScaler() function works by subtracting the mean of each feature and dividing it by the standard deviation of each feature. This ensures that the features are on the same scale and that no single feature dominates the clustering process.

The fit_transform() method of the StandardScaler() function fits the scaler to the data and then transforms the data.

Here , in my code, the scaler is fitted to the features AnnualIncome and SpendingScore in the DataFrame df and then the data is transformed.

Determining the optimal number of clusters k:

There is no one definitive way to determine the optimal number of clusters k. One common approach is to use the elbow method. The elbow method plots the sum of squared errors (SSE) for different values of k. The optimal number of clusters is the point where the SSE curve starts to bend.

Query 1:

Import libraries , read CSV file, and print the file

Clause:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df= pd.read_csv('mall.csv')
df
```

Result :

Out[4]:

	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

Query 2: Print the top 2 rows.

Clause:

df.head(2)

Result :

```
In [5]: df.head(2)
```

Out[5]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81

Query 3: Print the last 2 rows of the data frame.

Clause:

df.tail(2)

Result :

```
In [6]: df.tail(2)
```

Out[6]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
198	199	Male	32	137	18
199	200	Male	30	137	83

Query 4:

To get the shape of the DataFrame

Clause:

df.shape

Result :

```
In [5]: df.shape
Out[5]: (200, 5)
```

Query 5:

To get the information of the DataFrame

Clause:

df.info()

Result :

```
In [7]: df.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
```

Query 6:

To count the number of missing values in each column of a DataFrame

Clause:

df.isnull().sum()

Result:

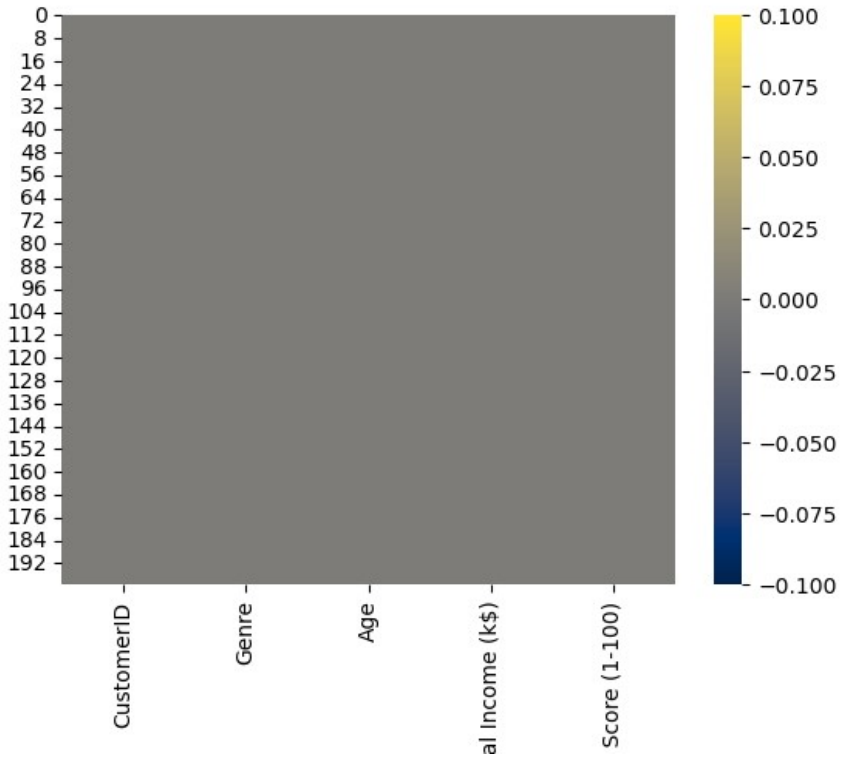
```
In [6]: df.isnull().sum()

Out[6]: CustomerID              0
        Genre                   0
        Age                     0
        Annual Income (k$)      0
        Spending Score (1-100)  0
        dtype: int64
```

Query 7 : Create a heatmap of the missing values in the data frame.

Clause:

sns.heatmap(df.isnull(),cmap='cividis')

Result:



Query 8 :

Clause:

```
df.dtypes.value_counts()
```

Result:

```
In [7]: df.dtypes.value_counts()

Out[7]: int64     4
        object    1
        dtype: int64
```

Query 9 :

Clause:

```
int_col=df.select_dtypes(include="int64").columns.tolist()
int_col
```

Result:

```
Out[8]: ['CustomerID', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)']
```

Query 10: Calculate the correlation coefficient between each pair of columns in the data frame and store it in the variable cor

To calculate and print a summary of statistical data for each column in the DataFrame

Clause:

df.describe()

Result:

```
In [9]: df.describe()
```

Out[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

Query 11:

Clause:

cat_col=df.select_dtypes(include="O").columns.tolist()
cat_col

Result:

```
In [10]: cat_col=df.select_dtypes(include="O").columns.tolist()
         cat_col

Out[10]: ['Genre']
```

Query 12:

Clause:

df.columns=df.columns.str.replace(" ","")
df.columns

Result:

```
In [11]: df.columns=df.columns.str.replace(" ","")
         df.columns

Out[11]: Index(['CustomerID', 'Genre', 'Age', 'AnnualIncome(k$)',
                'SpendingScore(1-100)'],
               dtype='object')
```

Query 13:

Clause:

```
df.columns=df.rename(columns={'AnnualIncome(k$)':'AnnualIncome',
                              'SpendingScore(1-100)':'SpendingScore',"Genre":"Gender"}).columns
df.columns
```

Result:

```
Out[12]: Index(['CustomerID', 'Gender', 'Age', 'AnnualIncome', 'SpendingScore'], dtype='object')
```

Query 14:

Clause:

```
df2=df.copy()
df.drop("CustomerID",axis=1,inplace=True)
```

Result:

Query 15:

Clause:

```
df.describe()
```

Result:

```
In [16]: df.describe()
```

Out[16]:

	Age	AnnualIncome	SpendingScore
count	200.000000	200.000000	200.000000
mean	38.850000	60.560000	50.200000
std	13.969007	26.264721	25.823522
min	18.000000	15.000000	1.000000
25%	28.750000	41.500000	34.750000
50%	36.000000	61.500000	50.000000
75%	49.000000	78.000000	73.000000
max	70.000000	137.000000	99.000000

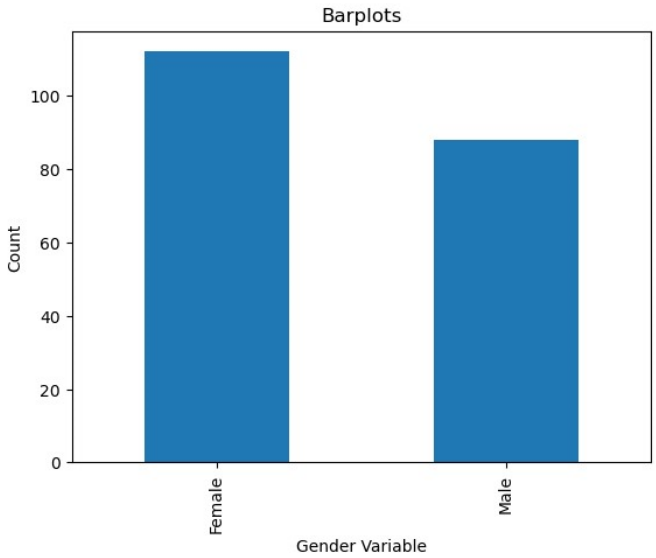
Query 16:

Clause:

```
df.Gender.value_counts().plot(kind="bar")
plt.title("Barplots")
plt.xlabel("Gender Variable")
```

```
plt.ylabel("Count")
```

Result:

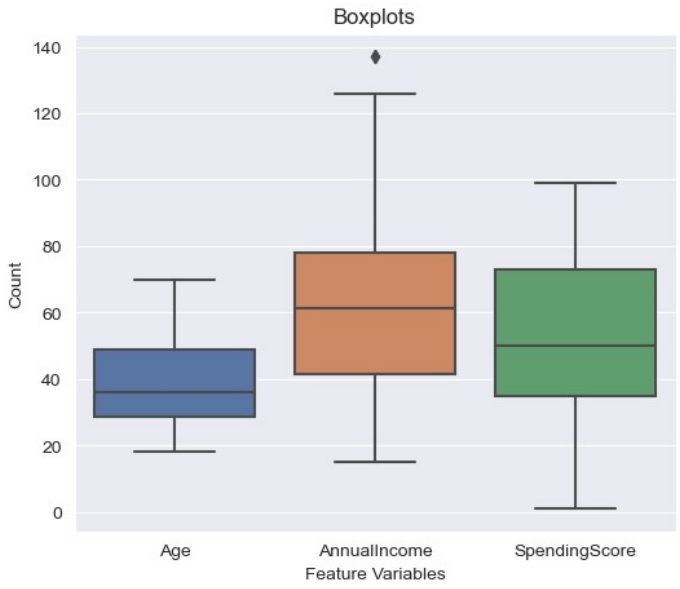


Query 17:

Clause:

```
sns.set({"figure.figsize":(6,5)})
sns.boxplot(df)
plt.title("Boxplots")
plt.xlabel("Feature Variables")
plt.ylabel("Count")
```

Result:

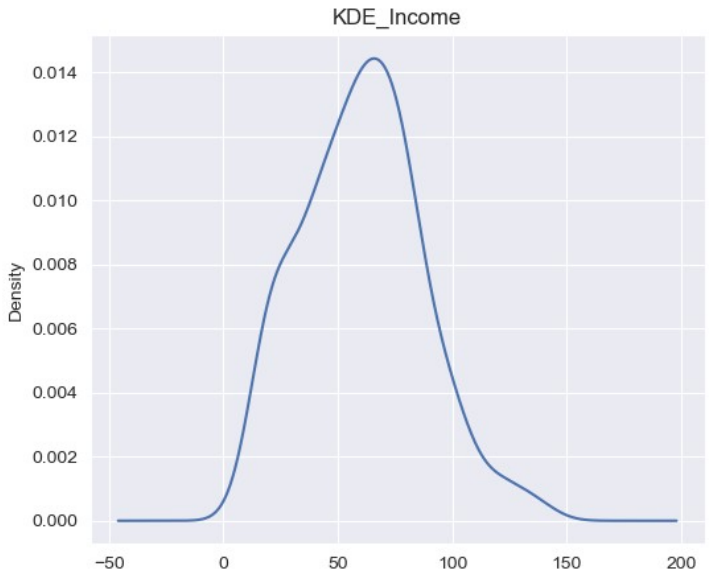


Query 18:

Clause:

```
df.AnnualIncome.plot(kind="kde")
plt.title('KDE_Income')
plt.show()
```

Result:

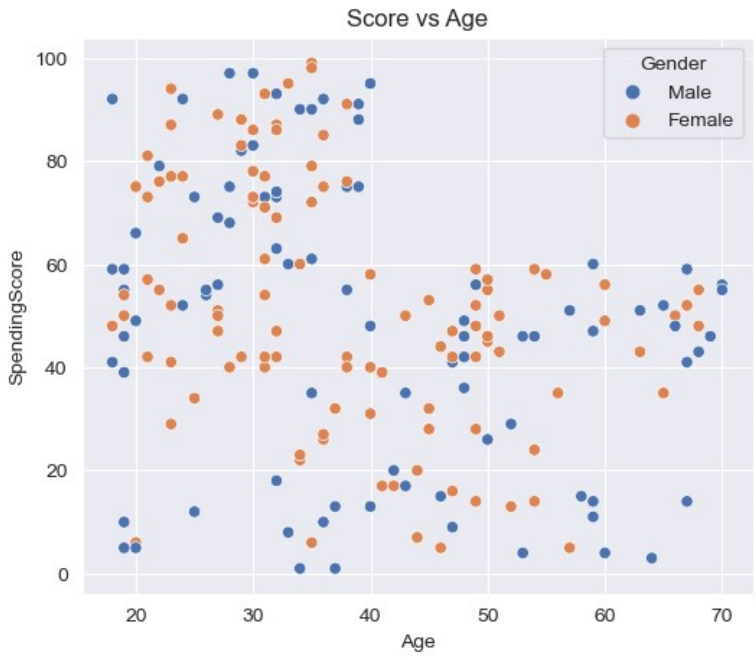


Query 19:

Clause:

```
sns.scatterplot(data=df,x='Age',y='SpendingScore',hue="Gender")
plt.title('Score vs Age')
plt.show()
```

Result:

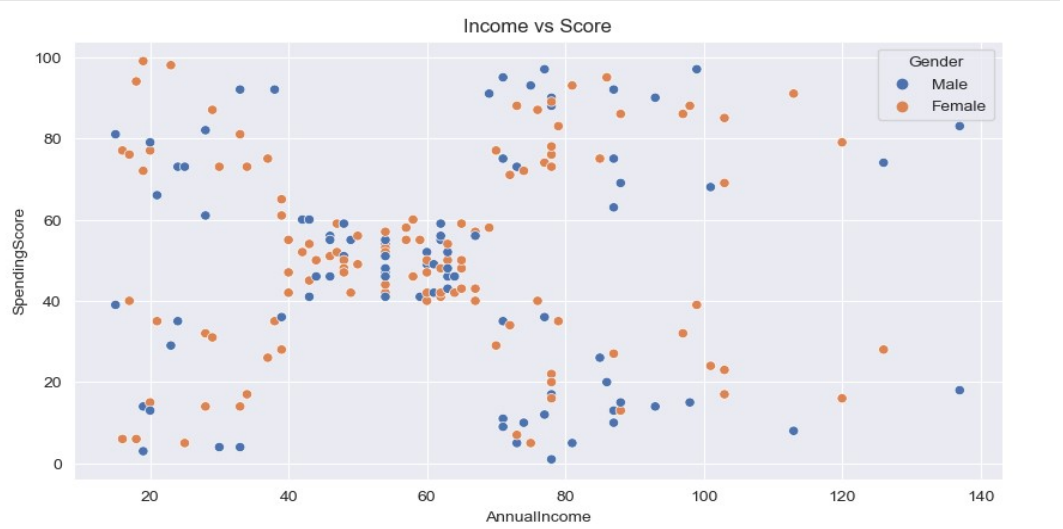


Query 20:

Clause:

```
sns.set({"figure.figsize":(10,5)})
sns.scatterplot(data=df,x='AnnualIncome',y='SpendingScore',hue="Gender")
plt.title('Income vs Score')
plt.show()
```

Result:



Query 21:

Clause:

```
df.Gender=np.where(df["Gender"]=="Male",1,0)
df.Gender
```

Result:

```
Out[24]: 0      1
         1      1
         2      0
         3      0
         4      0
               ..
         195    0
         196    0
         197    1
         198    1
         199    1
         Name: Gender, Length: 200, dtype: int32
```

Query 22:

Clause:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
s=StandardScaler()
s
```

Result:

```
In [28]: s

Out[28]: StandardScaler()
```

Query 23:

Clause:

```
s_val=s.fit_transform(df[["AnnualIncome","SpendingScore"]])
s_val
```

Result:

```
In [31]: s_val

Out[31]: array([[-1.73899919, -0.43480148],
                [-1.73899919,  1.19570407],
                [-1.70082976, -1.71591298],
                [-1.70082976,  1.04041783],
                [-1.66266033, -0.39597992],
                [-1.66266033,  1.00159627],
                [-1.62449091, -1.71591298],
                [-1.62449091,  1.70038436],
                [-1.58632148, -1.83237767],
                [-1.58632148,  0.84631002],
                [-1.58632148, -1.4053405 ],
```

Query 24:

Clause:

```
features=pd.DataFrame(s_val,columns=df.columns[2:4].tolist())
features
```

Result:

Out[32]:

	AnnualIncome	SpendingScore
0	-1.738999	-0.434801
1	-1.738999	1.195704
2	-1.700830	-1.715913
3	-1.700830	1.040418
4	-1.662660	-0.395980
...	...	...
195	2.268791	1.118061
196	2.497807	-0.861839
197	2.497807	0.923953
198	2.917671	-1.250054
199	2.917671	1.273347

200 rows × 2 columns

Query 25:

Clause:

```
er=[]
for k in range(1,20):
    kmeans=KMeans(n_clusters=k,random_state=101)
    kmeans.fit(features)
    er.append(kmeans.inertia_)

er[:6]
```

Result:

```
er[:6]
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warn
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMean
leak on Windows with MKL, when there are less chunks than available threads. You can avoid it
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warn
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMean
leak on Windows with MKL, when there are less chunks than available threads. You can avoid it
```

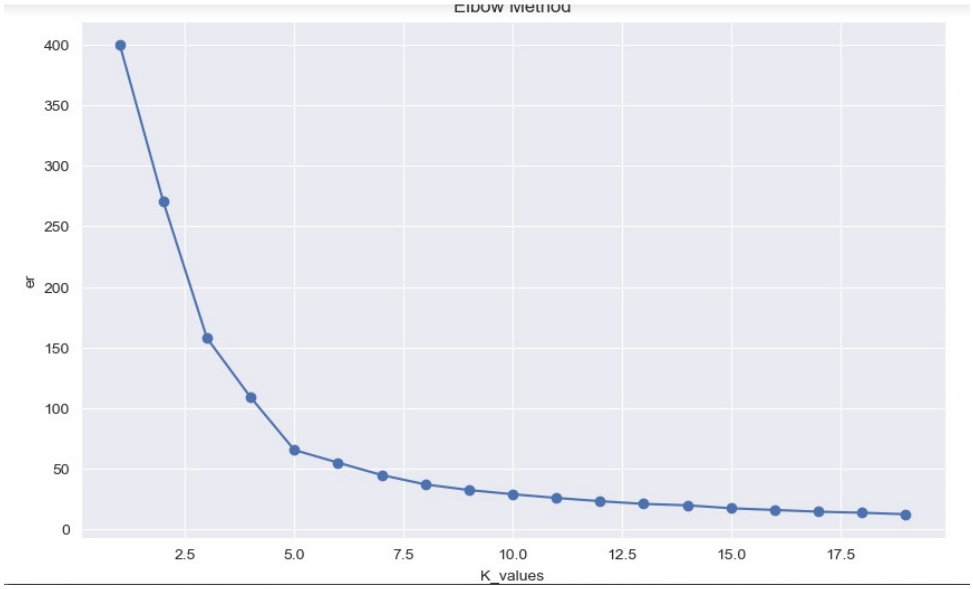
Query 26:

Clause:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,20),er,marker="o")

plt.xlabel('K_values')
plt.ylabel('er')
plt.title('Elbow Method')
```

plt.show()

Result:



Query 27:

Clause:

kval=5

kmeans=KMeans(n_clusters=kval,max_iter=10,random_state=101)
kmeans.fit(features)

Result:

```
kmeans=KMeans(n_clusters=kval,max_iter=10,random_state=101)
kmeans.fit(features)
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default v
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is know
ak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting
ble OMP_NUM_THREADS=1.
  warnings.warn(
```

```
0]: KMeans(max_iter=10, n_clusters=5, random_state=101)
```

Query 28:

Clause:

*labels=kmeans.labels_*
labels

Result:

```
In [58]: labels=kmeans.labels_
         labels

Out[58]: 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, 0,
                4, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1, 2, 1, 2, 1,
                0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
                2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
                2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
                2, 1])
```

Query 29:

Clause:

```
ccentroids=kmeans.cluster_centers_
centroids
```

Result:

```
Out[59]: array([[-0.20091257, -0.02645617],
                [ 0.99158305,  1.23950275],
                [ 1.05500302, -1.28443907],
                [-1.32954532,  1.13217788],
                [-1.30751869, -1.13696536]])
```

Query 30:

Clause:

```
e=kmeans.inertia_
e
```

Result:

```
In [53]: e=kmeans.inertia_
         e

Out[53]: 65.56840815571681
```

Query 31:

Clause:

```
itr=kmeans.n_iter_
itr
```

Result:

```
In [54]:  itr=kmeans.n_iter_
          itr

Out[54]: 4
```

Query 32:

Clause:

```
new_features=features.assign(clusters=pd.DataFrame(labels))
new_features
```

Result:

Out[60]:

	AnnualIncome	Spending Score	clusters
0	-1.738999	-0.434801	4
1	-1.738999	1.195704	3
2	-1.700830	-1.715913	4
3	-1.700830	1.040418	3
4	-1.662660	-0.395980	4
...	...	...	...
195	2.268791	1.118061	1
196	2.497807	-0.861839	2
197	2.497807	0.923953	1
198	2.917671	-1.250054	2
199	2.917671	1.273347	1

200 rows × 3 columns

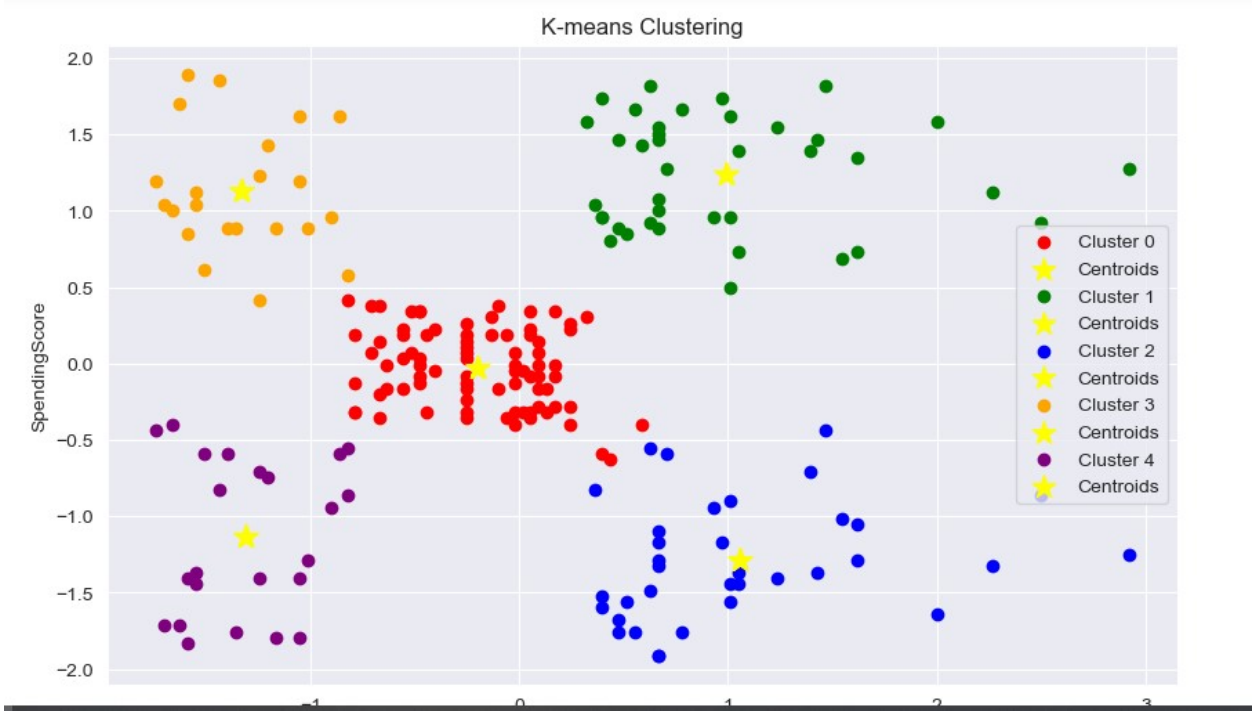
Query 33:

Clause:

```
plt.figure(figsize=(10, 6))

clr=["red","green","blue","orange","purple"]
for cluster_num in range(kval):
    plt.scatter(x=new_features[new_features.clusters == cluster_num]['AnnualIncome'],
            y= new_features[new_features.clusters ==
cluster_num]['SpendingScore'],marker='o',c=clr[cluster_num]
            ,label=f'Cluster {cluster_num}')
    plt.scatter(x=centroids[:, 0], y=centroids[:, 1],
          c='yellow', marker='*', s=150, label='Centroids')

plt.xlabel('AnnualIncome')
plt.ylabel('SpendingScore')
plt.title('K-means Clustering')
```

```
plt.legend()
plt.show()
```

Result:
The following plot shows the cluster results for the Mall Customers dataset. The different clusters are colored differently.



Query 34:

Clause:

```
cust1=df[df["labels"]==1] print('Number of customer in 1st group=', len(cust1)) print('They are -', cust1["CustomerID"].values) print("-----------------------------------------")
cust2=df[df["labels"]==2] print('Number of customer in 2nd group=', len(cust2)) print('They are -', cust2["CustomerID"].values) print("-----------------------------------------")
cust3=df[df["labels"]==0] print('Number of customer in 3rd group=', len(cust3)) print('They are -', cust3["CustomerID"].values) print("-----------------------------------------")
cust4=df[df["labels"]==3] print('Number of customer in 4th group=', len(cust4)) print('They are -', cust4["CustomerID"].values) print("-----------------------------------------")
cust5=df[df["labels"]==4] print('Number of customer in 5th group=', len(cust5)) print('They are -', cust5["CustomerID"].values) print("-----------------------------------------")
```

Result:

```
Number of customer in 1st group= 22
They are - [129 131 135 137 139 141 145 149 151 153 155 157 159 163 165 167 169 171
 173 175 177 179]
-----------------------------------------------
Number of customer in 2nd group= 32
They are - [124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158
 160 162 164 166 168 170 172 174 176 178 180 182 184 186]
-----------------------------------------------
Number of customer in 3rd group= 27
They are - [ 41  47  51  54  55  57  58  60  61  63  64  65  68  71  73  74  75  81
  83  87  91 103 107 109 110 111 117]
-----------------------------------------------
Number of customer in 4th group= 26
They are - [ 59  62  66  69  70  76  79  85  88  89  92  95  96  98 100 101 104 106
 112 114 115 116 121 125 133 143]
-----------------------------------------------
Number of customer in 5th group= 13
They are - [ 3  7  9 11 13 15 19 23 25 31 33 35 37]
-----------------------------------------------
```

Characteristics and Common Traits:

Finally, the entire clustering analysis is given below:

Cluster 0 (Red):

Characteristics: Average annual income and an average spending score.

Description: These customers are well-balanced in terms of their income and spending. They are stable and reliable, but not necessarily the most profitable.

Cluster 1 (Green):

Characteristics: Higher annual income and higher spending score.

Description: These are the most profitable customers, likely high-value customers willing to spend generously on your products or services.

Cluster 2 (Blue):

Characteristics: Higher annual income and lower spending score.

Description: Less profitable than Segment 1 but still valuable. They might be careful spenders, looking for good deals.

Cluster 3 (Orange):

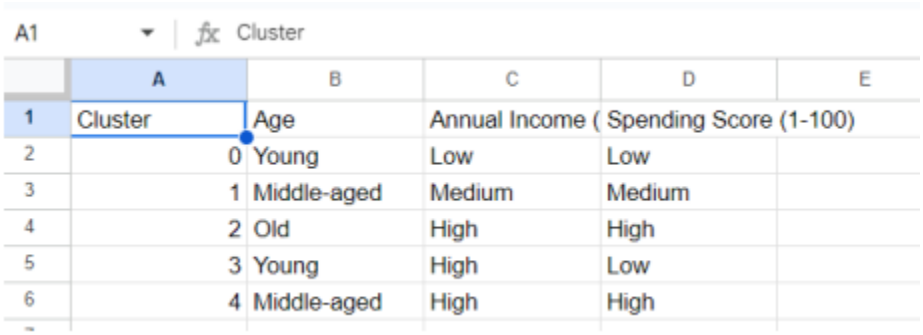
Characteristics: Lower annual income and higher spending score.

Description: These are the least profitable customers. They may be financially struggling, spending more than they can afford.

Cluster 4 (Purple):

Characteristics: Lower annual income and lower spending score.

Description: Balanced and careful customers in terms of income and spending. Likely budget-conscious customers seeking ways to save money



	A	B	C	D	E
1	Cluster	Age	Annual Income (	Spending Score (1-100)	
2	0	Young	Low	Low	
3	1	Middle-aged	Medium	Medium	
4	2	Old	High	High	
5	3	Young	High	Low	
6	4	Middle-aged	High	High	

.

Query 35:

Clause:

```
score=metrics.silhouette_score(features,kmeans.labels_)
print("Silhouette_Score Coefficient : {:.2f}".format(score))
```

Result:

```
Silhouette_Score Coefficient : 0.55
```