Assignment No. 7

Problem Statement: Apply the K-Means clustering algorithm to segment customers based on their annual income and spending score using the Mall Customers dataset.

Objective:

1. To group mall customers into distinct clusters based on their behavior using unsupervised learning.
2. To identify customer segments that can be used for targeted marketing strategies.
3. To understand the working of the K-Means algorithm and apply it to real-world data.

Prerequisite :

1. Basic knowledge of Python programming
2. Familiarity with pandas and NumPy libraries
3. Understanding of data preprocessing techniques
4. Knowledge of K-Means Clustering algorithm
5. Ability to visualize data using matplotlib or seaborn
6. Awareness of clustering evaluation metrics (silhouette score)

Theory :

K-Means Clustering is an unsupervised machine learning algorithm used to divide a dataset into groups, or *clusters*, of similar data points. The goal is to group data in such a way that points within the same cluster are very similar to each other, and points in different clusters are quite different. This is especially helpful when we don't have labeled data and want to explore natural patterns or groupings.

Working:

1. **Choosing the Number of Clusters (K)**
   The first step is deciding how many clusters (K) you want. This is usually a number that the analyst sets based on their understanding of the problem or by using methods like the Elbow Method or Silhouette Score to find the best value.

2. **Placing the Centroids**
   K random points are chosen from the data to act as the initial **centroids** (think of these as the centers of your clusters).

3. **Assigning Points to Clusters**
   Each data point is then assigned to the closest centroid. The "closeness" is measured by calculating the **Euclidean distance** between the data point and each centroid.

4. **Recalculating Centroids**
   After all data points have been assigned to a cluster, the centroids are updated. Each new centroid is calculated as the **average (mean)** of all the points in its cluster.

5. **Repeating the Process**
   Steps 3 and 4 are repeated until the centroids don't change much anymore or the assignments stop changing. This means the algorithm has *converged*, and the clustering is done.

The strength of K-Means lies in its simplicity. By grouping similar items and reducing the distance within clusters, it effectively uncovers the hidden patterns in the data. It's especially useful for segmenting customers, grouping search results, organizing inventory, etc.

Use Cases in Real Life

1. **Customer Segmentation**: Businesses often use K-Means to segment their customers based on purchase behavior, age, income, or spending patterns
   .
2. **Document Classification**: Articles or emails can be grouped into categories based on topics or word usage.

3. **Image Compression**: It can group similar colors to reduce the size of image files without losing much quality.

4. **City Planning**: Government departments use it to group areas by population, income, traffic levels, etc., to plan better infrastructure.

Advantages

1. Easy to understand and implement
2. Works well on large datasets
3. Efficient in terms of computation
4. Gives clear clusters if the data is well-separated

Limitations

1. You need to know the value of K in advance
2. It doesn't perform well with non-spherical clusters or data with different densities
3. Sensitive to outliers and noise
4. If the initial centroids are poorly chosen, results can vary (though K-Means++ helps with better initial centroid selection)

Key Concepts

1. **Euclidean Distance**: Measures how far a data point is from the centroid (as the crow flies).

2. **Centroid**: The average position of all the points in a cluster.

3. **Inertia**: A measure of how tightly the data points are grouped in each cluster (lower is better).

Code & Output

```
import numpy as np
import pandas as pd
df= pd.read_csv("C:/Users/dnyan/ML Assignments/Dataset/Mall_Customers.csv")
```

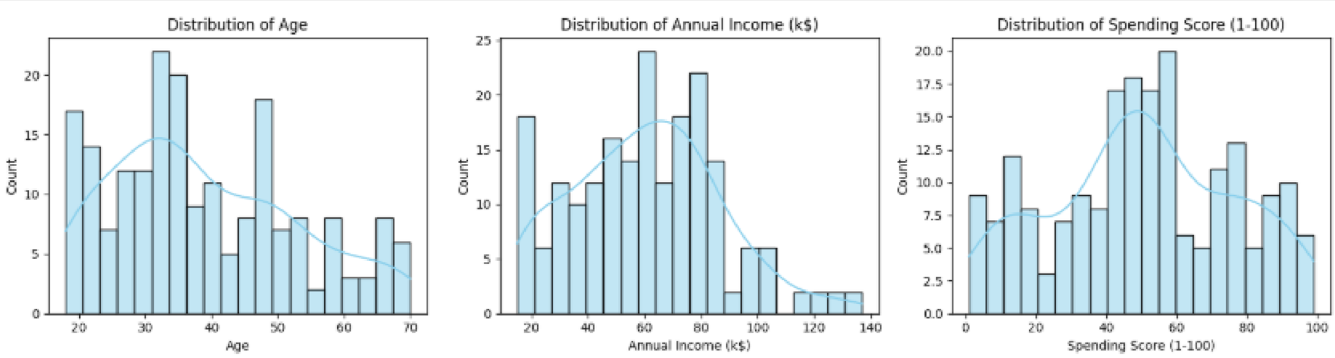
```
print(df.info())
print(df.describe())

# Check missing values
print(df.isnull().sum())
```

```
<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$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
None
       CustomerID         Age  Annual Income (k$)  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
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

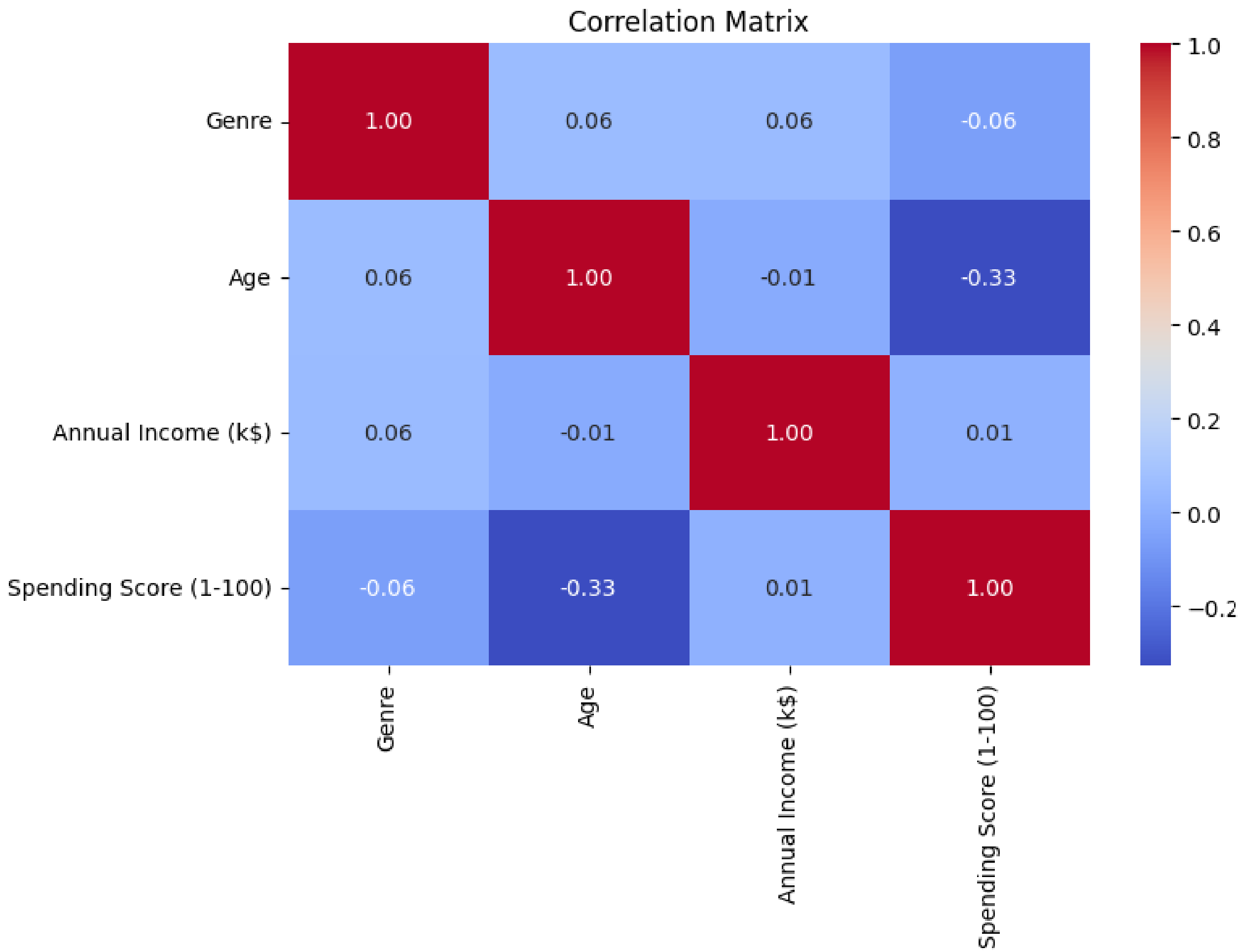
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Strip any accidental whitespace in column names
df.columns = df.columns.str.strip()

# Display column names to verify
print("Columns:", df.columns.tolist())

# Drop CustomerID column
if 'CustomerID' in df.columns:
    df.drop('CustomerID', axis=1, inplace=True)

# Encode 'Genre' column (Male/Female) using Label Encoding
le = LabelEncoder()
df['Genre'] = le.fit_transform(df['Genre'])  # Male = 1, Female = 0 (usually)

# Feature scaling using StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Convert scaled data back to DataFrame for ease
df_scaled = pd.DataFrame(scaled_data, columns=df.columns)

# Show the first few rows of the preprocessed data
print("\nPreprocessed Data Sample:")
print(df_scaled.head())
```

```
Columns: ['Genre', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)']

Preprocessed Data Sample:
      Genre       Age  Annual Income (k$)  Spending Score (1-100)
0  1.128152 -1.424569           -1.738999               -0.434801
1  1.128152 -1.281035           -1.738999                1.195704
2 -0.886405 -1.352802           -1.700830               -1.715913
3 -0.886405 -1.137502           -1.700830                1.040418
4 -0.886405 -0.563369           -1.662660               -0.395980
```

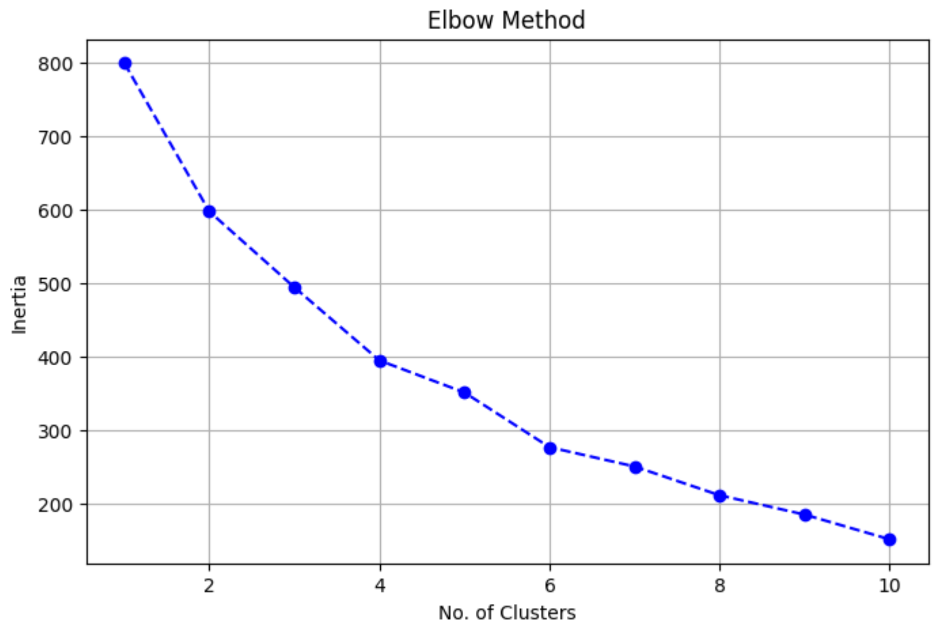
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(15,4))
for i, col in enumerate(['Age', 'Annual Income (k$)', 'Spending Score (1-100)']):
    plt.subplot(1, 3, i+1)
    sns.histplot(df[col], bins=20, kde=True, color='skyblue')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



```python
plt.figure(figsize=(8,5))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

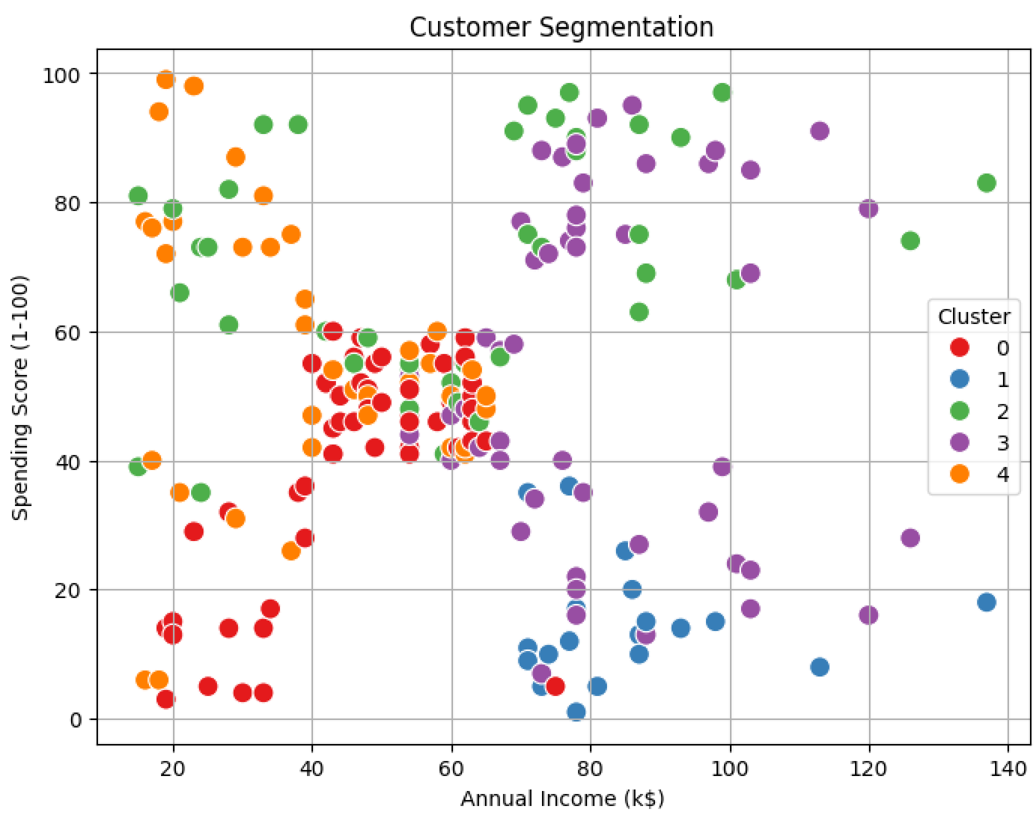



```python
from sklearn.cluster import KMeans

inertia = []
K = range(1,11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8,5))
plt.plot(K, inertia, 'bo--')
plt.xlabel('No. of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid()
plt.show()
```



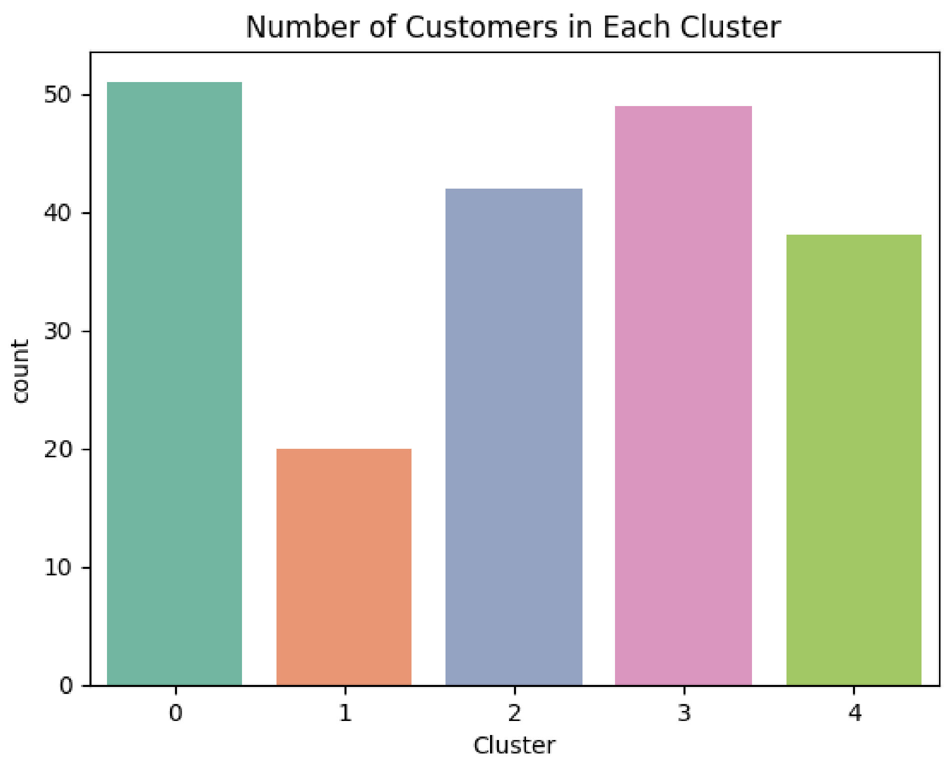

```
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)
```

```
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='Annual Income (k$)', y='Spending Score (1-100)',
                hue='Cluster', palette='Set1', s=100)
plt.title("Customer Segmentation")
plt.grid()
plt.show()
```



```
sns.countplot(x='Cluster', data=df, palette='Set2')
plt.title("Number of Customers in Each Cluster")
plt.show()
```

```
C:\Users\dnyan\AppData\Local\Temp\ipykernel_10332\3177132749.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
hue` and set `legend=False` for the same effect.

  sns.countplot(x='Cluster', data=df, palette='Set2')
```




```
from sklearn.metrics import silhouette_score

score = silhouette_score(scaled_data, df['Cluster'])
print(f"Silhouette Score: {score:.3f}")
```

```
Silhouette Score: 0.272
```

Github : https://github.com/dnyaneshwardhere/ML

Conclusion:

K-Means clustering effectively grouped mall customers based on their spending habits and income levels. It provided clear insights into different customer segments, which can be valuable for targeted marketing and strategic business decisions. The algorithm proved to be efficient, easy to implement, and interpretable, making it suitable for real-world customer segmentation tasks.