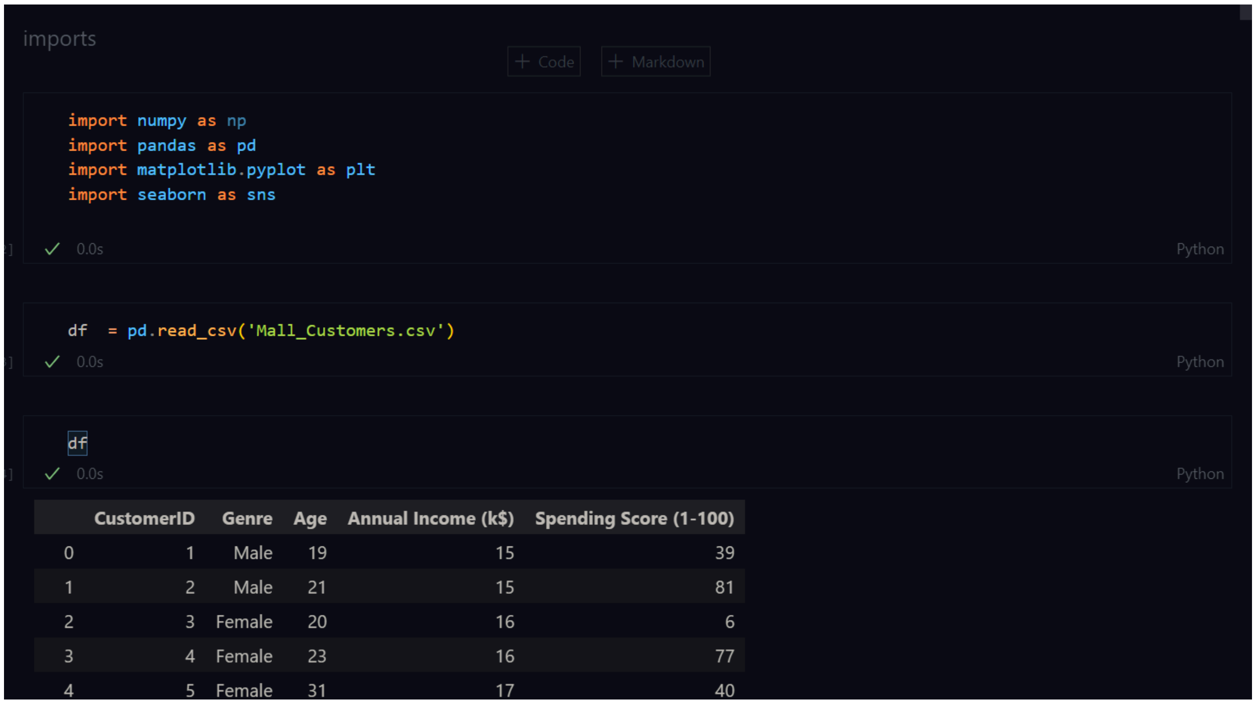
# Assignment -10

**-aditya verma**

**Dataset**
[https://drive.google.com/file/d/1ObiAGjo4DQzOvbpkE4I7DAlBcTh0SkXn/view?usp=classroom_web&authuser=0](https://drive.google.com/file/d/1ObiAGjo4DQzOvbpkE4I7DAlBcTh0SkXn/view?usp=classroom_web&authuser=0)

**Collab link**
[https://colab.research.google.com/drive/16OHppQEAaSSZEFVC1tGpCZWtJV66gwo4?usp=sharing](https://colab.research.google.com/drive/16OHppQEAaSSZEFVC1tGpCZWtJV66gwo4?usp=sharing)

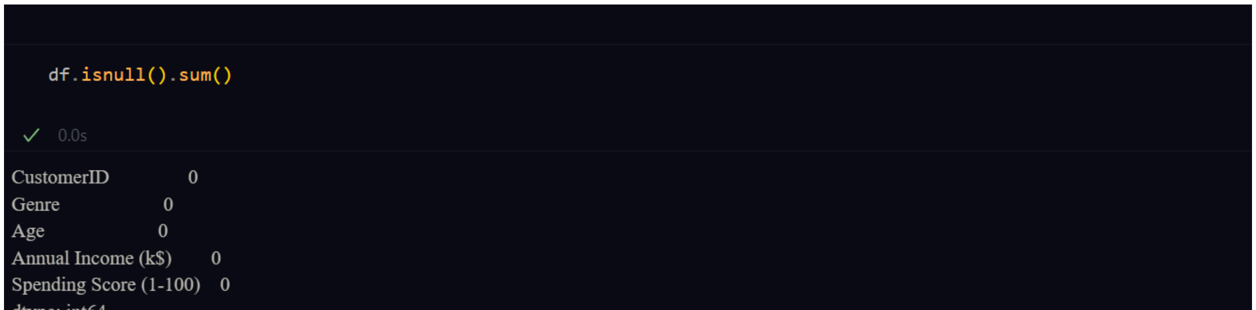


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df  = pd.read_csv('Mall_Customers.csv')
```

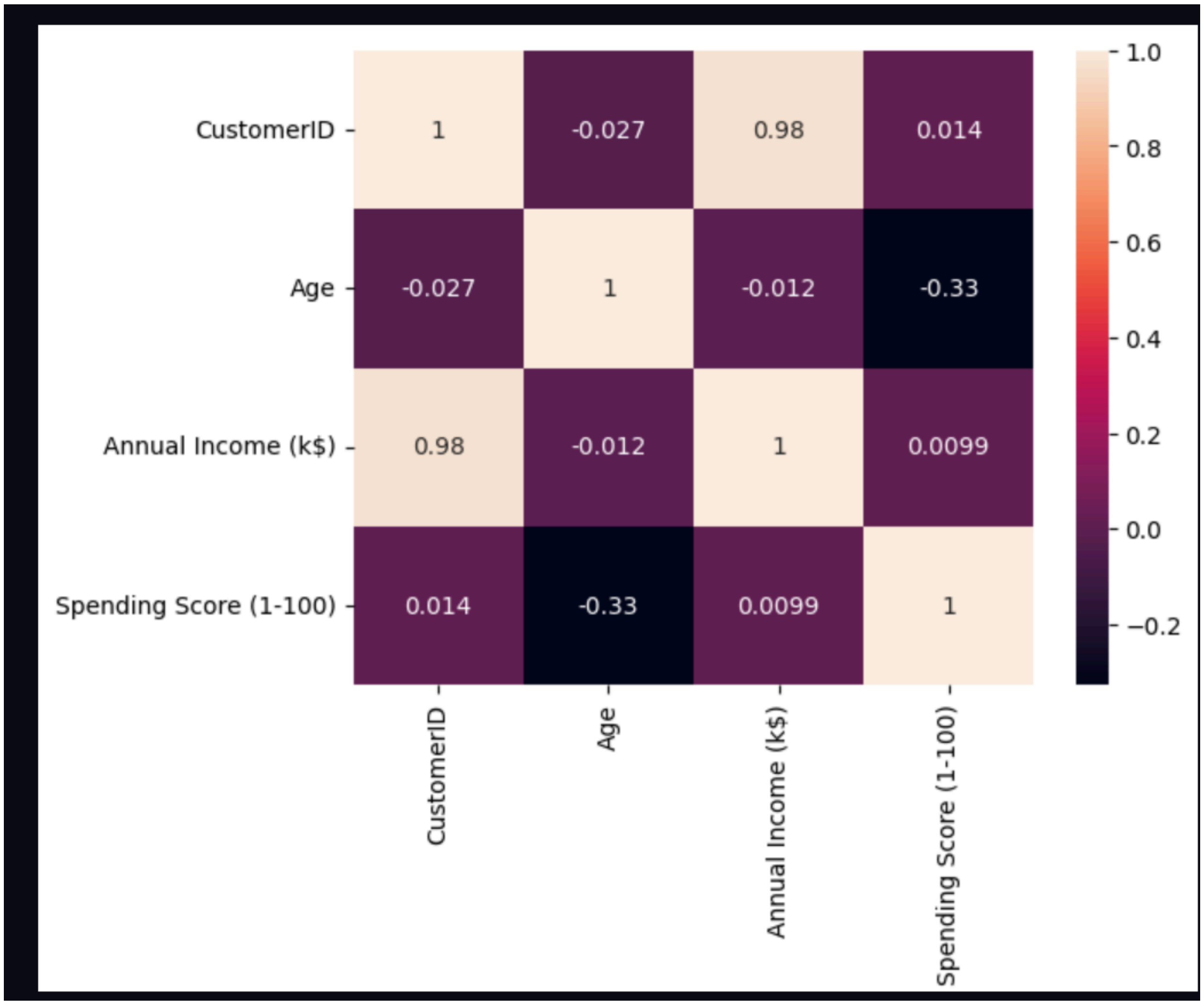
```
df
```

| | 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 |



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

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



**Problem**
**1: Customer Segmentation using K-Means**

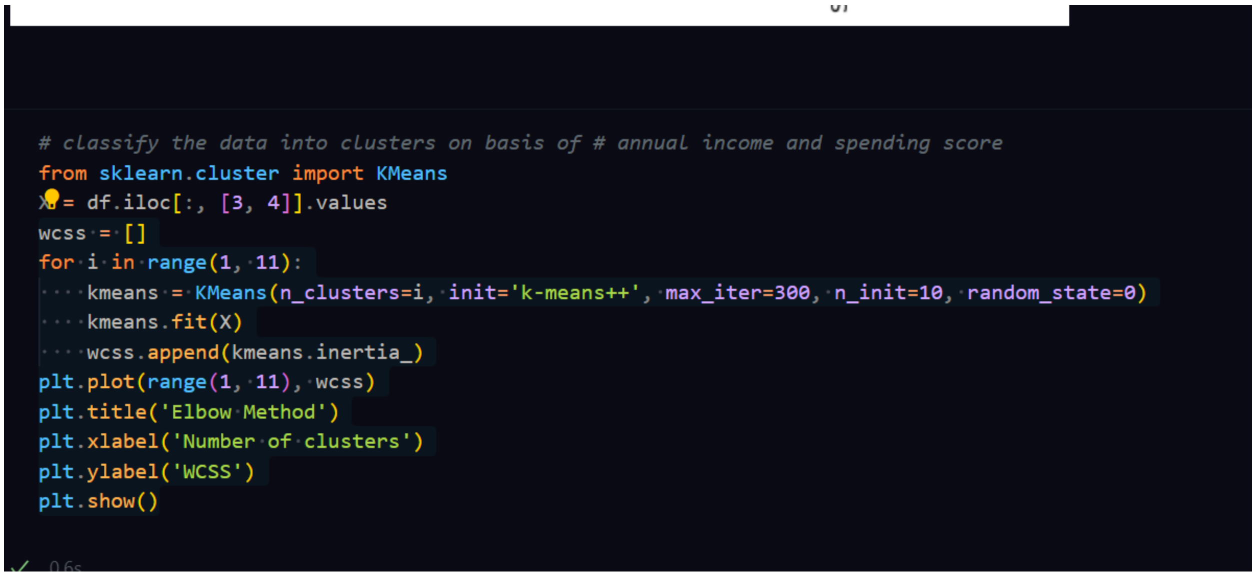
## Objective

**: Segment mall customers into distinct groups based on their shopping behavior.**

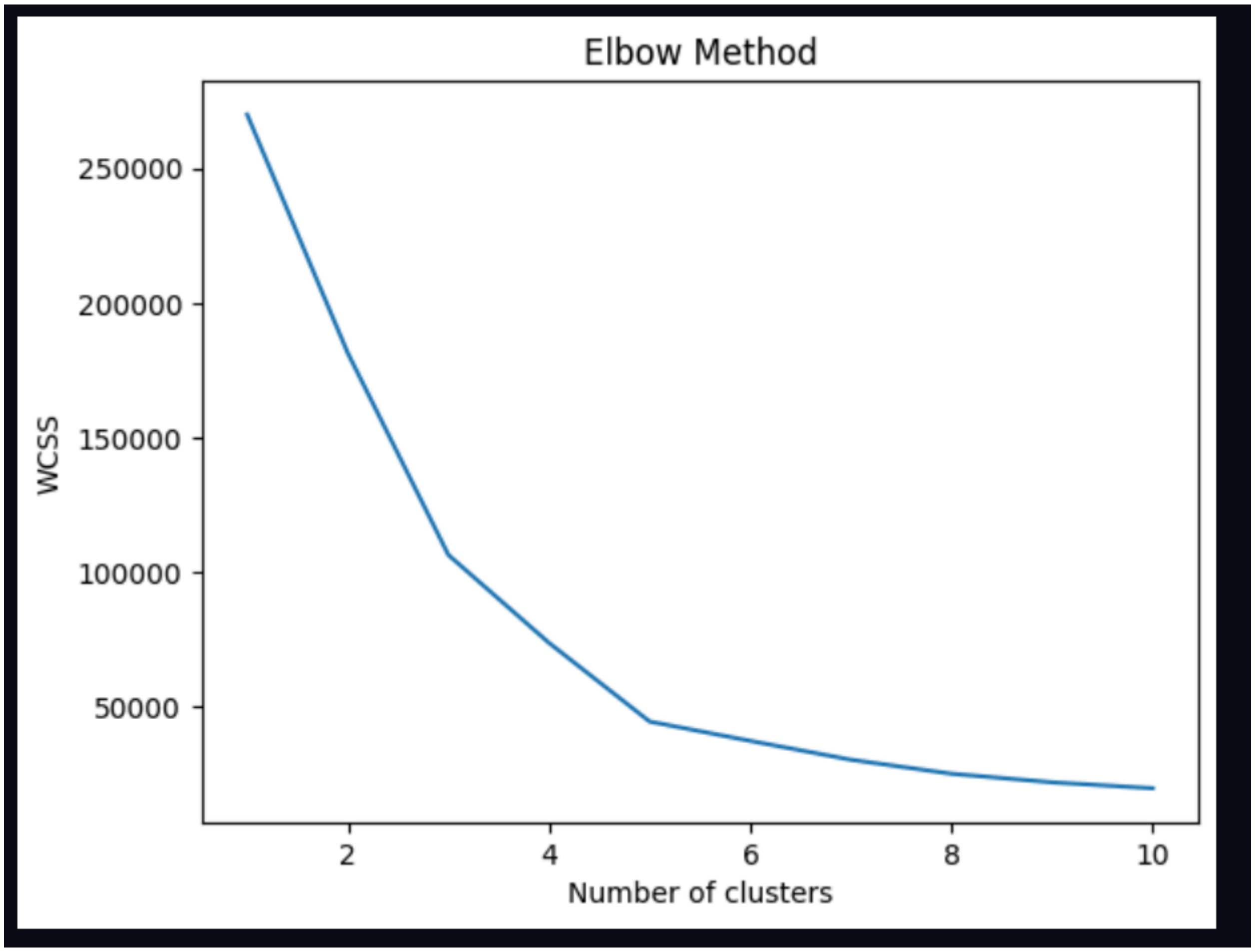
## Instructions

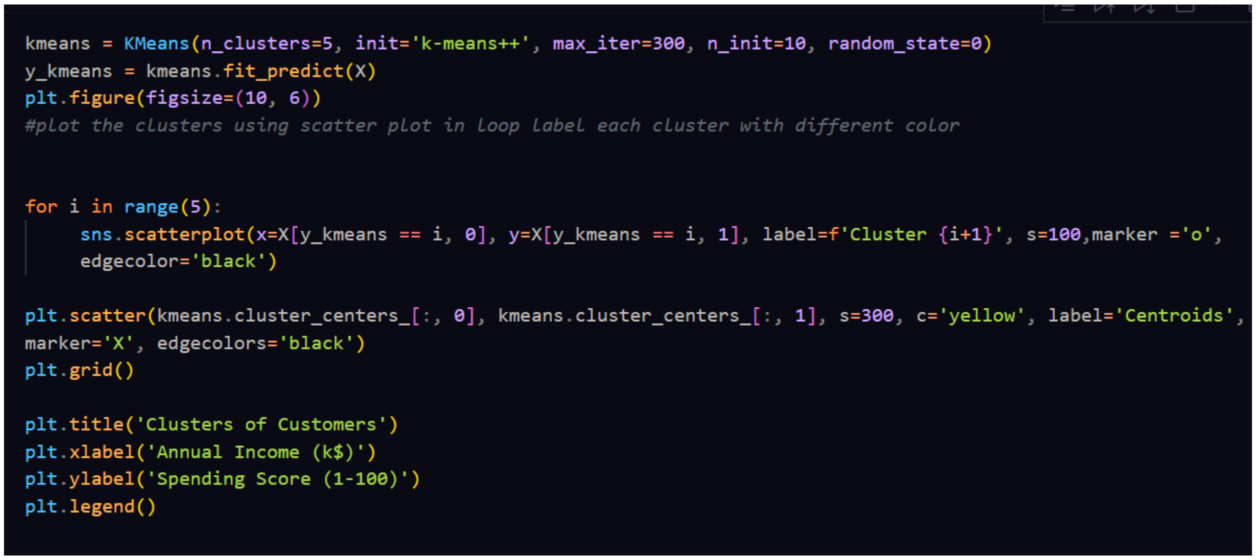
**:1.Load the dataset and handle any missing or irrelevant values.**

**2.Select relevant features for clustering (e.g., Annual Income (k$) and Spending Score(1-100)).**
**3.Normalize or scale the data appropriately.**
**4.Use the Elbow Method to determine the optimal number of clusters.**
**5.Apply K-Means clustering and visualize the clusters.**
**6. Label each customer with their cluster group and analyze the characteristics of each cluster.**

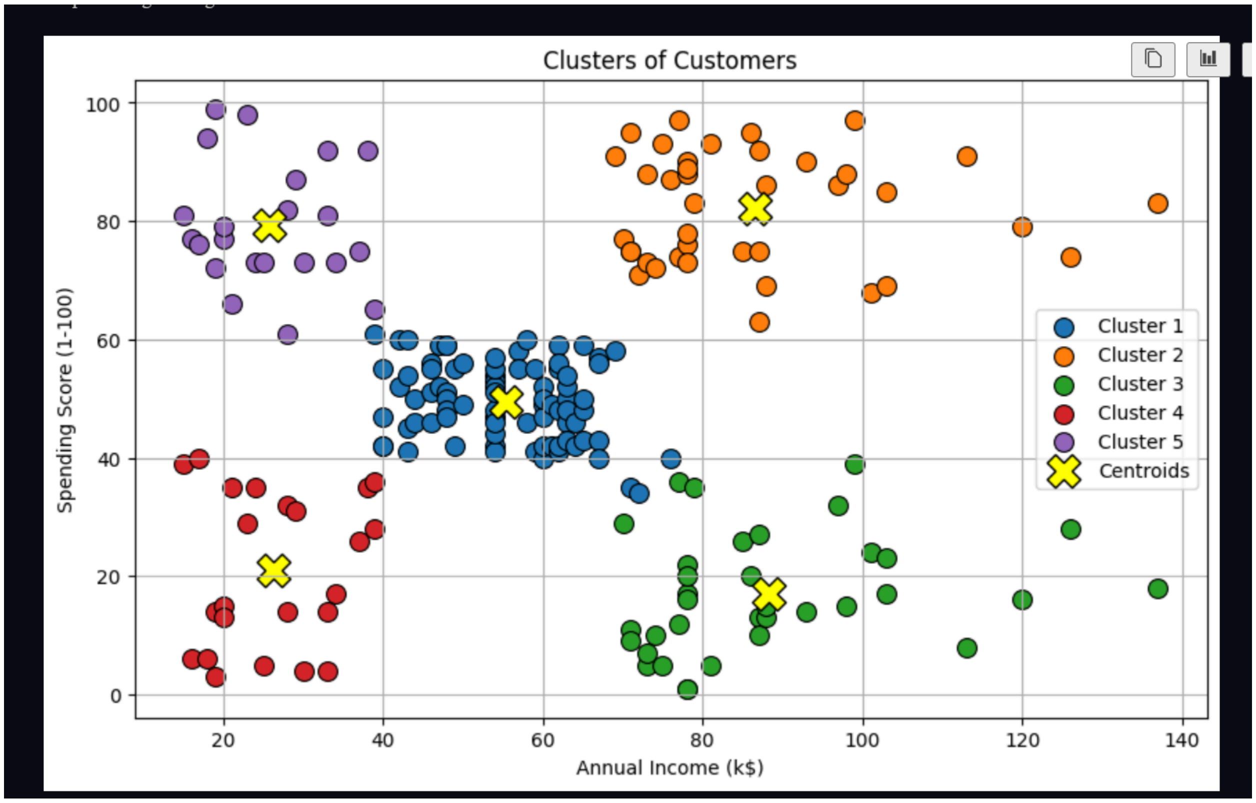


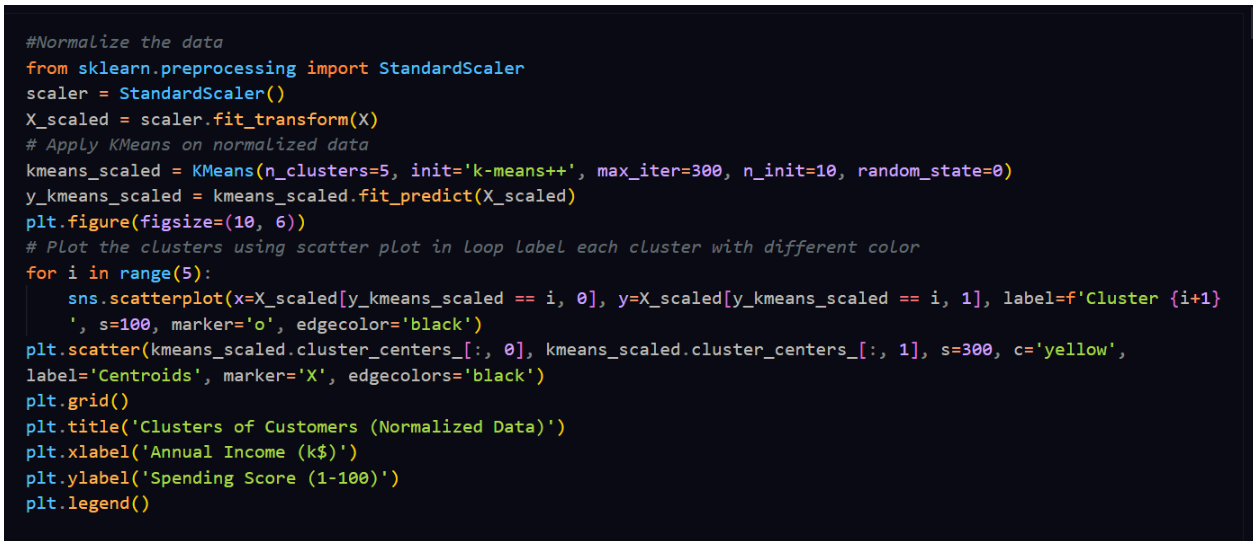
```
# classify the data into clusters on basis of # annual income and spending score
from sklearn.cluster import KMeans
X = df.iloc[:, [3, 4]].values
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



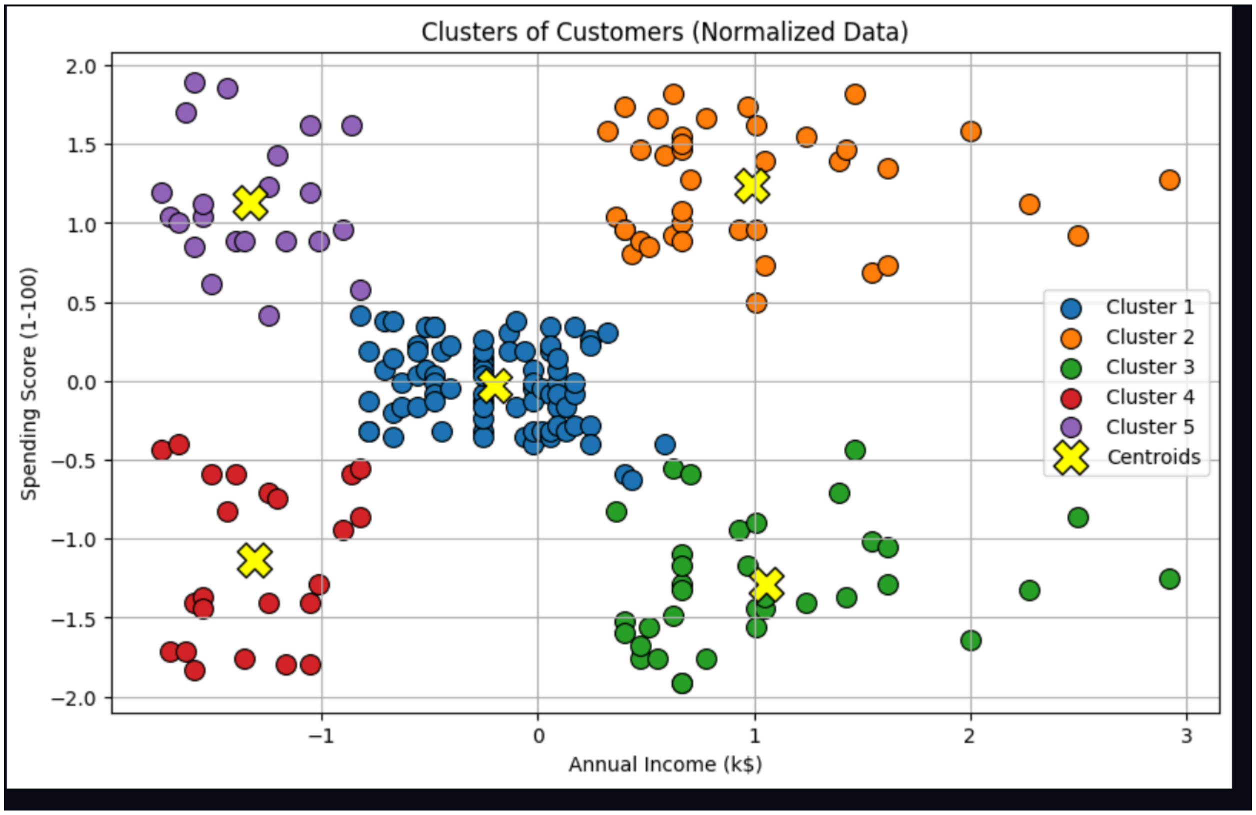


```
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(X)
plt.figure(figsize=(10, 6))
#plot the clusters using scatter plot in loop label each cluster with different color


for i in range(5):
    sns.scatterplot(x=X[y_kmeans == i, 0], y=X[y_kmeans == i, 1], label=f'Cluster {i+1}', s=100,marker ='o',
    edgecolor='black')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label='Centroids',
marker='X', edgecolors='black')
plt.grid()

plt.title('Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
```





```
#Normalize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply KMeans on normalized data
kmeans_scaled = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_scaled = kmeans_scaled.fit_predict(X_scaled)
plt.figure(figsize=(10, 6))
# Plot the clusters using scatter plot in loop label each cluster with different color
for i in range(5):
    sns.scatterplot(x=X_scaled[y_kmeans_scaled == i, 0], y=X_scaled[y_kmeans_scaled == i, 1], label=f'Cluster {i+1}
    ', s=100, marker='o', edgecolor='black')
plt.scatter(kmeans_scaled.cluster_centers_[:, 0], kmeans_scaled.cluster_centers_[:, 1], s=300, c='yellow',
label='Centroids', marker='X', edgecolors='black')
plt.grid()
plt.title('Clusters of Customers (Normalized Data)')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
```



**Questions**
**:How many clusters does the Elbow Method suggest?**
**Ans - 5**

**Describe each cluster's characteristics in terms of spending habits.**
**Ans -**

| Cluster | Income Level | Spending Behavior | Marketing Strategy |
|---|---|---|---|
| Cluster 1 | Average | Moderate-Low | Emphasize value, budget deals |
| Cluster 2 | High | High | Target premium/luxury items |

| | | | |
|---|---|---|---|
| Cluster3 | High | Low | Offer loyalty savings, value bundles |
| Cluster 4 | Low | Low | Focus on affordability, essentials |
| Cluster 5 | Low | High | Financing plans, impulse buy channels |

## What insights can mall management derive from this segmentation?

### targeted Marketing & Promotions

- **Cluster 2 (high-income, high-spenders): Promote premium brands, exclusive events, and loyalty perks. Use VIP invitations or premium loyalty tiers to reinforce engagement.**

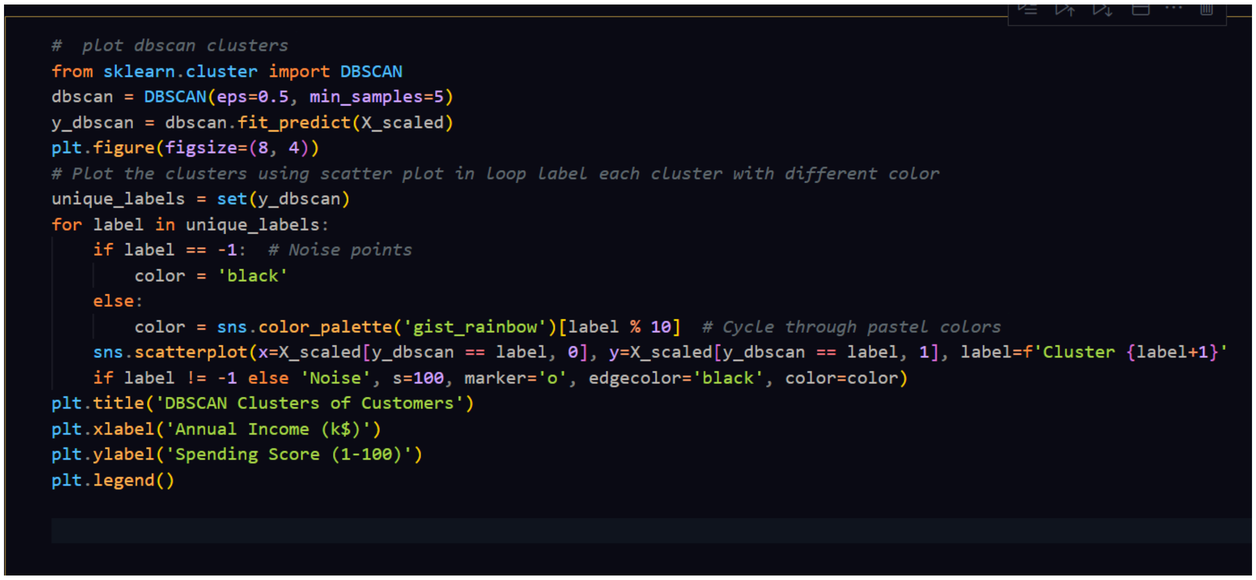
- **Cluster 5 (low-income, high-spenders): Offer payment flexibility (e.g., EMIs), mid-tier discount campaigns, or financing options to support their aspirational spending behaviors.**

Cluster 1 (average income, moderate-spenders): Focus on value-based loyalty programs, in-mall events, and seasonal promotions to encourage repeat visits. medium.com+2medium.com+2github.com+2

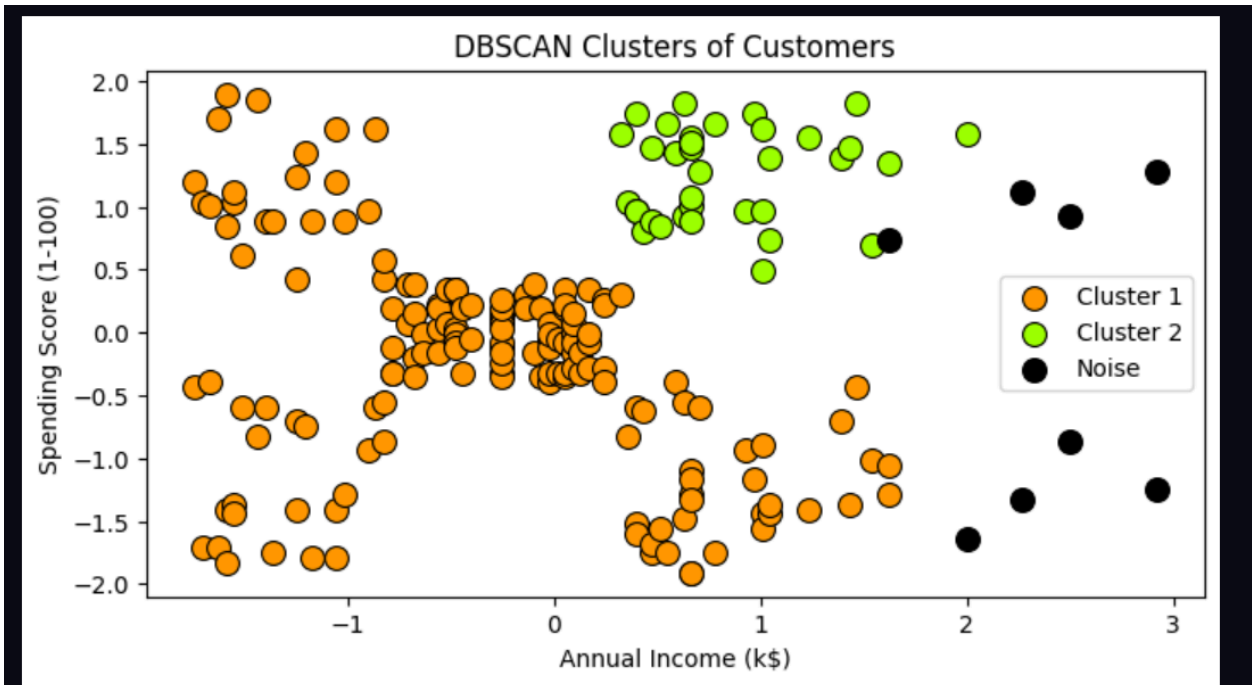
Cluster 4 (low-income, low-spenders): Lower priority for investment. Instead, reassess the marketing for this group—identify if location, pricing, or competition is to blame

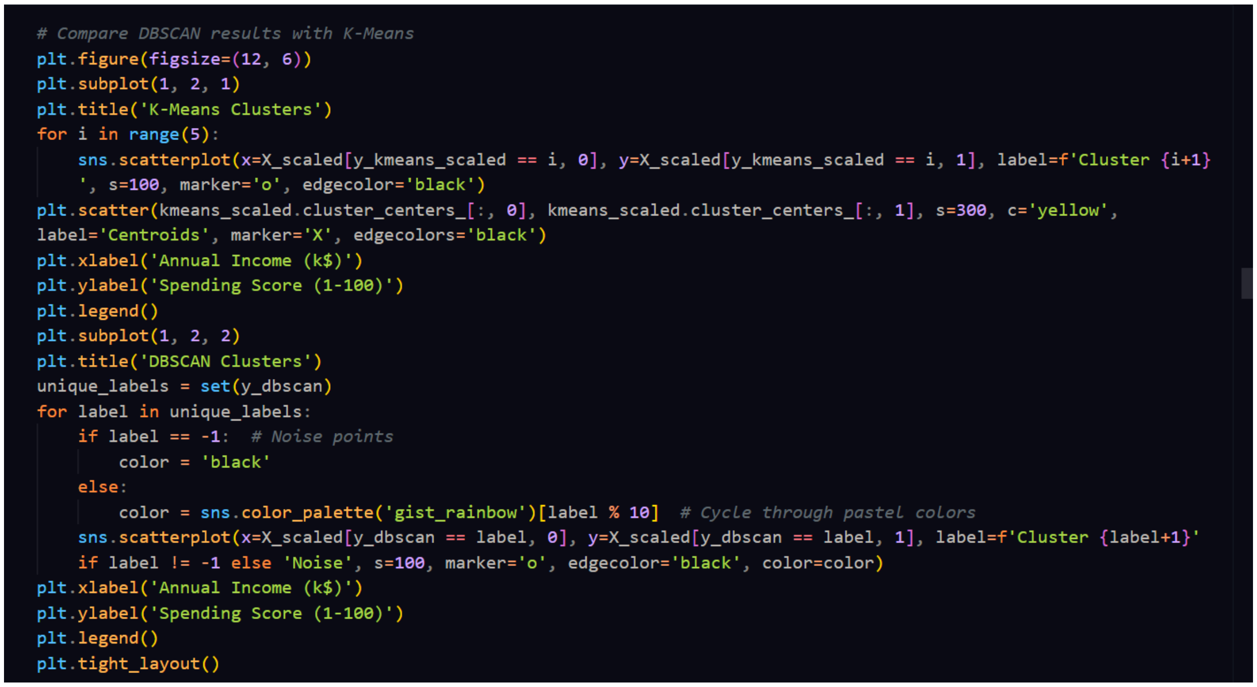
**Problem**
**2: Density-Based Clustering with DBSCAN**
**Objective**
**: Discover customer clusters of varying density using DBSCAN.**
**Instructions**
**1.Use the same features as in Assignment 1.**
**2.Normalize the data.**
**3.Use DBSCAN to perform clustering. Experiment with different eps and min_samplesvalues**
**4.Visualize the clusters and identify outliers.**

## 5.Compare DBSCAN results with K-Means.

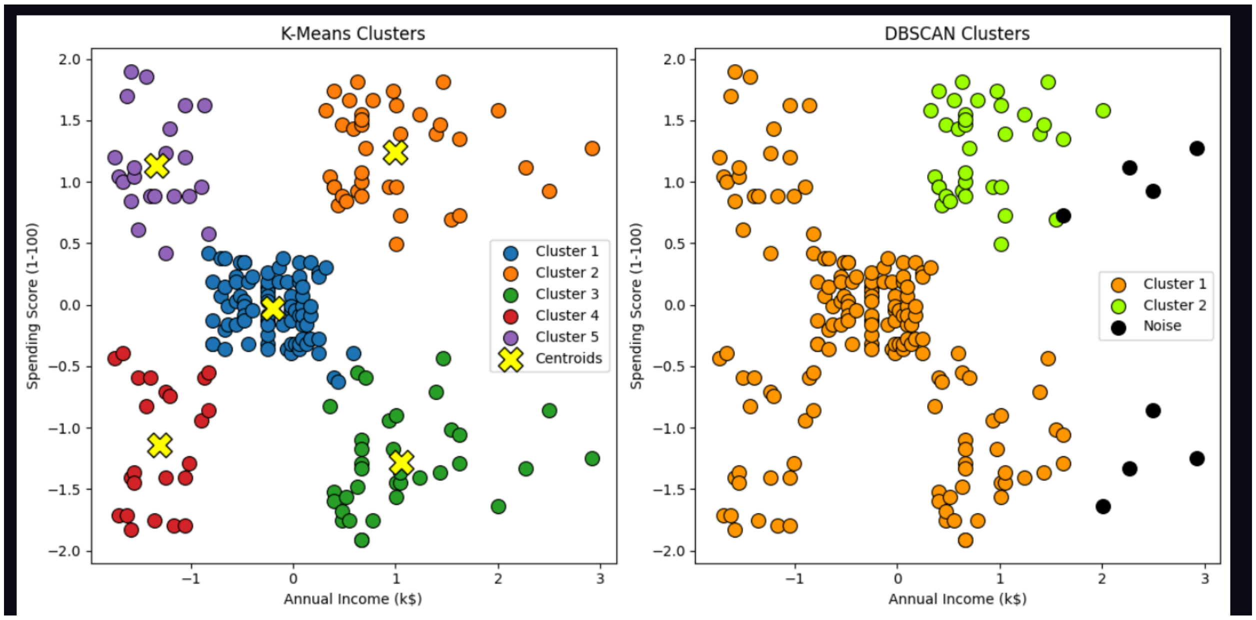


```
#  plot dbscan clusters
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_dbscan = dbscan.fit_predict(X_scaled)
plt.figure(figsize=(8, 4))
# Plot the clusters using scatter plot in loop label each cluster with different color
unique_labels = set(y_dbscan)
for label in unique_labels:
    if label == -1:  # Noise points
        color = 'black'
    else:
        color = sns.color_palette('gist_rainbow')[label % 10]  # Cycle through pastel colors
    sns.scatterplot(x=X_scaled[y_dbscan == label, 0], y=X_scaled[y_dbscan == label, 1], label=f'Cluster {label+1}'
    if label != -1 else 'Noise', s=100, marker='o', edgecolor='black', color=color)
plt.title('DBSCAN Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
```





```python
# Compare DBSCAN results with K-Means
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('K-Means Clusters')
for i in range(5):
    sns.scatterplot(x=X_scaled[y_kmeans_scaled == i, 0], y=X_scaled[y_kmeans_scaled == i, 1], label=f'Cluster {i+1}
    ', s=100, marker='o', edgecolor='black')
plt.scatter(kmeans_scaled.cluster_centers_[:, 0], kmeans_scaled.cluster_centers_[:, 1], s=300, c='yellow',
label='Centroids', marker='X', edgecolors='black')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.subplot(1, 2, 2)
plt.title('DBSCAN Clusters')
unique_labels = set(y_dbscan)
for label in unique_labels:
    if label == -1:  # Noise points
        color = 'black'
    else:
        color = sns.color_palette('gist_rainbow')[label % 10]  # Cycle through pastel colors
    sns.scatterplot(x=X_scaled[y_dbscan == label, 0], y=X_scaled[y_dbscan == label, 1], label=f'Cluster {label+1}'
    if label != -1 else 'Noise', s=100, marker='o', edgecolor='black', color=color)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.tight_layout()
```



## Questions
:
## What are the advantages of DBSCAN in this context?

**Ans -**No need to predefine the number of clusters – unlike K-means, DBSCAN automatically determines how many clusters exist via density thresholds.
Detects arbitrarily shaped clusters – it effectively handles irregular group shapes, capturing dense spending-income pockets that aren't spherical

Identifies outliers (noise) – customers that don't belong to any clear segment are marked as noise, which helps spot uncommon or fraud-prone behaviors .
Robust to outliers – because noisy points are separated, core clusters aren't skewed by extreme data

## How many core points and outliers were detected

From the DBSCAN cluster plot:

- **Core points**: Most data points in dense groups (around average income/moderate, high-income/high-spending, and high-income/low-spending) are core points.

- **Outliers (black)**: There appear to be about **7–10 noisy points** on the right side (e.g., those with very high income and spending). These are not assigned to any cluster, flagged as outliers.

8 points are labeled as noise.

## Does DBSCAN reveal any patterns that K-Means missed?

**Noise detection**: DBSCAN flagged unique individuals at the high-income extreme as noise—something K-Means forced into a cluster, potentially skewing centroids.

**Irregular cluster shapes**: DBSCAN split the high-income space into two density-based segments:

- One cluster is **high-income, low-spending**

- Another is **high-income, high-spending**
  Although only two clusters appear, the shapes are more reflective of real density rather than artificially separated spherical blobs.

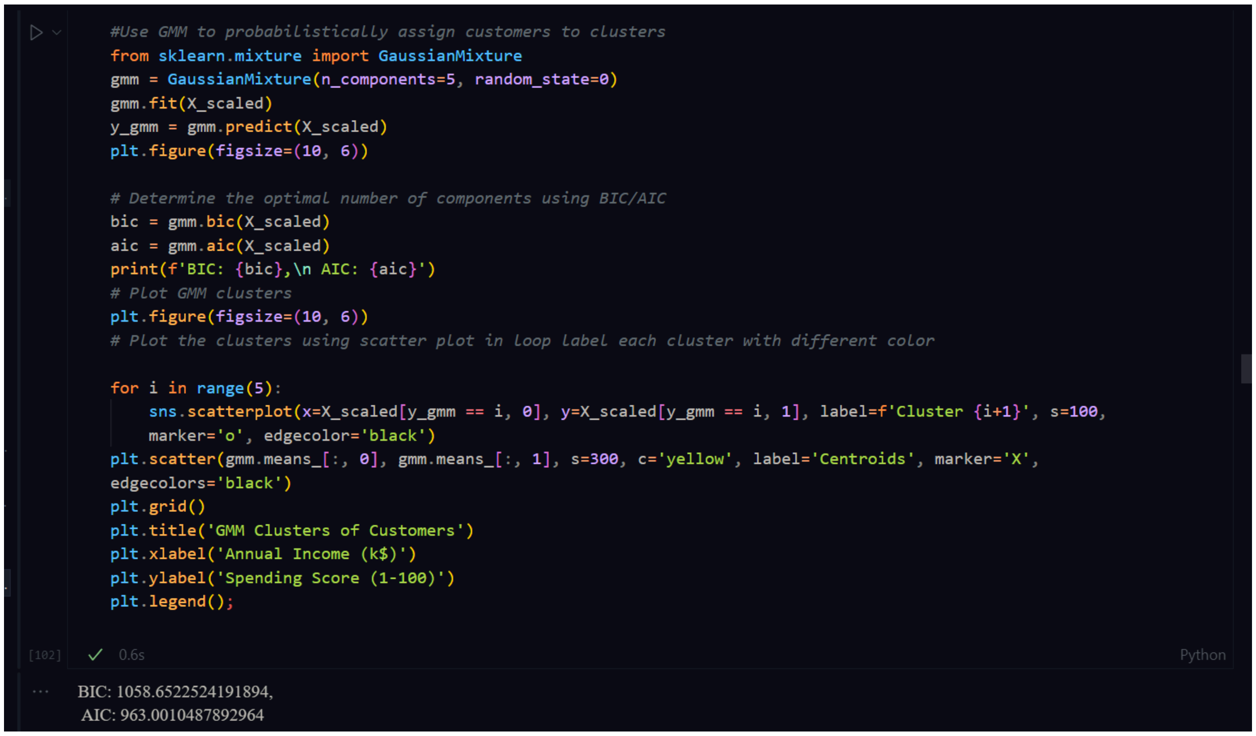
**No forced assignments**: Individuals with unusual income/spend combinations (like ultra-wealthy low-spenders) are not forced into ill-fitting clusters, preserving segmentation integrity

## Problem3: Gaussian Mixture Models (GMM) for Soft Clustering

### Objective: Use GMM to probabilistically assign customers to clusters.

### 1.Reuse the preprocessed data from Assignment 1.
### 2.Fit a Gaussian Mixture Model to the dataset.
### 3.Determine the optimal number of components using BIC/AIC.

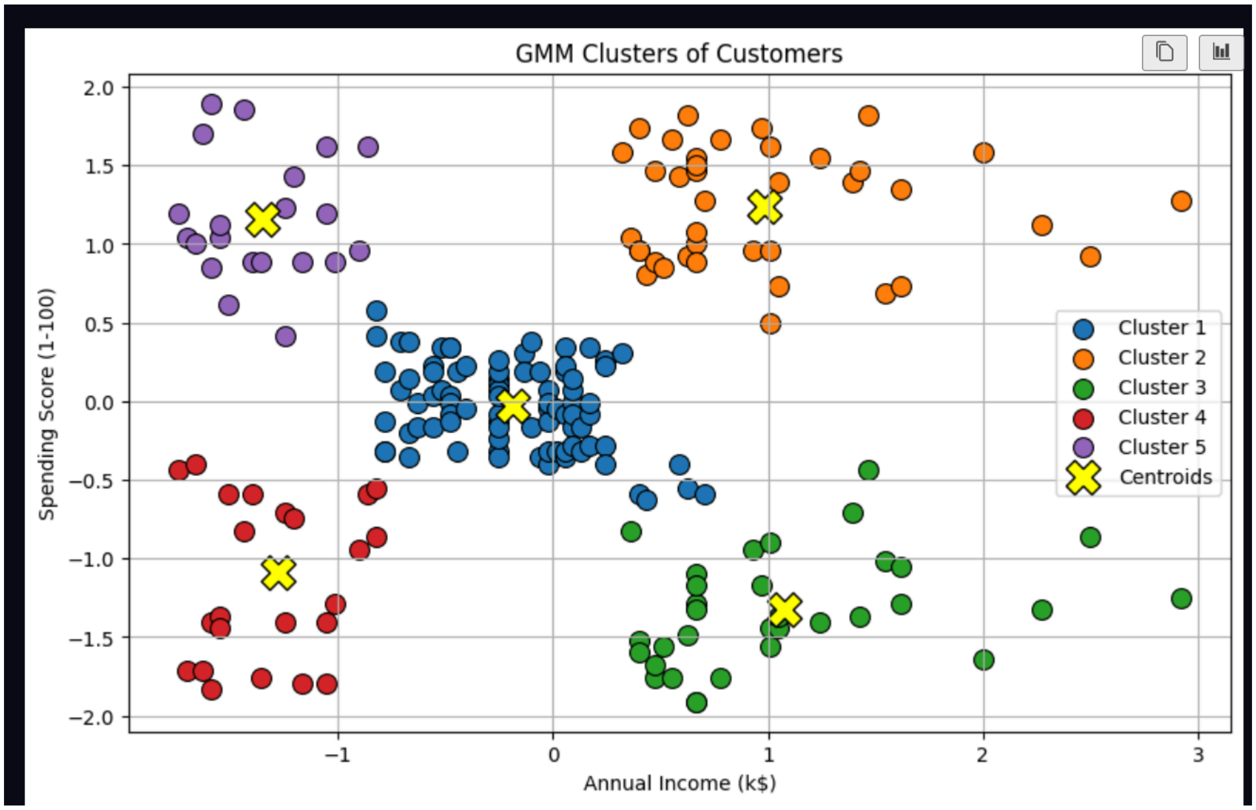
**4.Visualize the probability-based clusters.**
**5.Compare GMM results with K-Means and DBSCAN.**

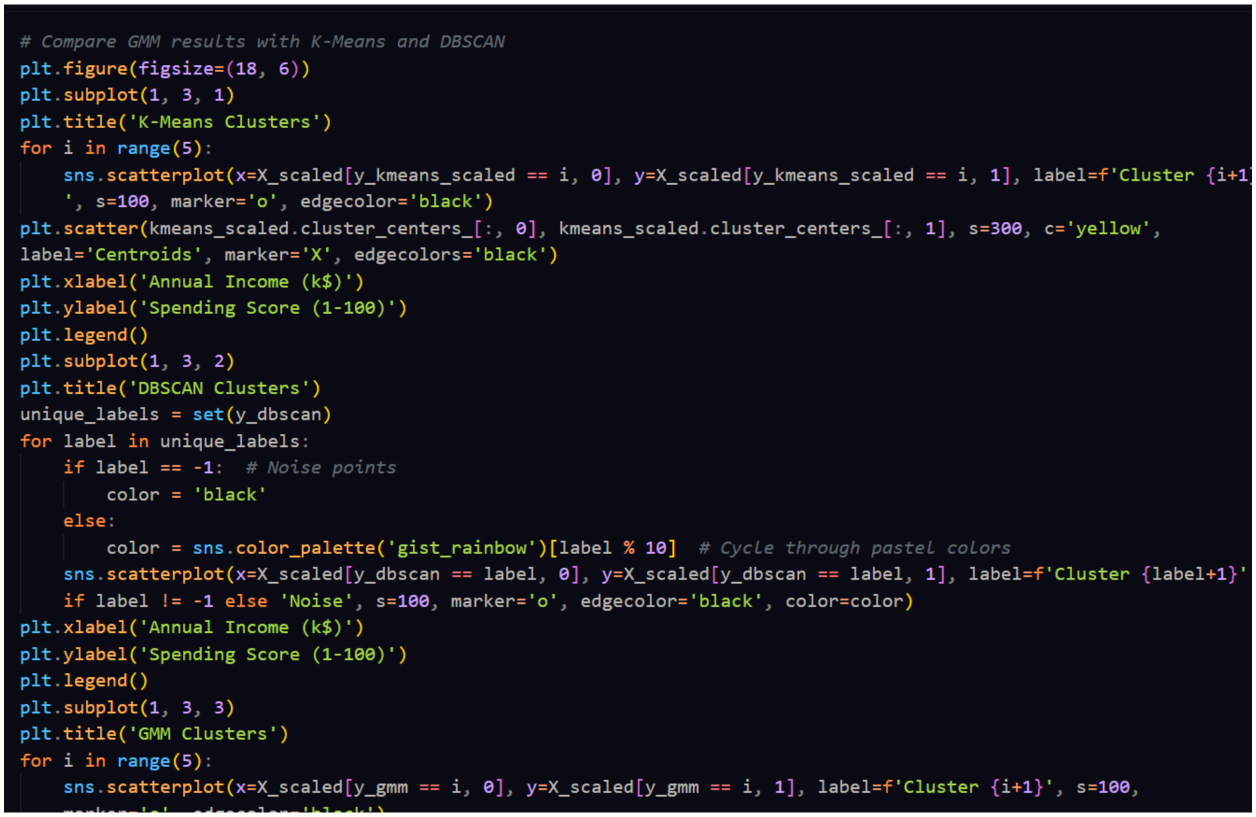


```python
#Use GMM to probabilistically assign customers to clusters
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=5, random_state=0)
gmm.fit(X_scaled)
y_gmm = gmm.predict(X_scaled)
plt.figure(figsize=(10, 6))

# Determine the optimal number of components using BIC/AIC
bic = gmm.bic(X_scaled)
aic = gmm.aic(X_scaled)
print(f'BIC: {bic},\n AIC: {aic}')
# Plot GMM clusters
plt.figure(figsize=(10, 6))
# Plot the clusters using scatter plot in loop label each cluster with different color

for i in range(5):
    sns.scatterplot(x=X_scaled[y_gmm == i, 0], y=X_scaled[y_gmm == i, 1], label=f'Cluster {i+1}', s=100,
    marker='o', edgecolor='black')
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], s=300, c='yellow', label='Centroids', marker='X',
edgecolors='black')
plt.grid()
plt.title('GMM Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend();
```

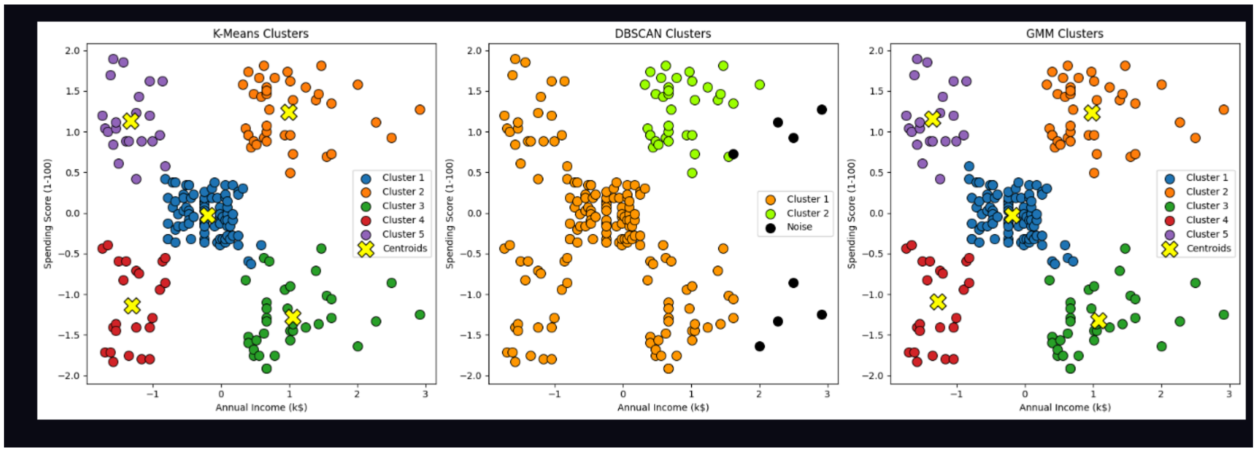
[102] ✓ 0.6s Python

```
BIC: 1058.6522524191894,
 AIC: 963.0010487892964
```





```python
# Compare GMM results with K-Means and DBSCAN
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.title('K-Means Clusters')
for i in range(5):
    sns.scatterplot(x=X_scaled[y_kmeans_scaled == i, 0], y=X_scaled[y_kmeans_scaled == i, 1], label=f'Cluster {i+1
    ', s=100, marker='o', edgecolor='black')
plt.scatter(kmeans_scaled.cluster_centers_[:, 0], kmeans_scaled.cluster_centers_[:, 1], s=300, c='yellow',
label='Centroids', marker='X', edgecolors='black')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.subplot(1, 3, 2)
plt.title('DBSCAN Clusters')
unique_labels = set(y_dbscan)
for label in unique_labels:
    if label == -1:  # Noise points
        color = 'black'
    else:
        color = sns.color_palette('gist_rainbow')[label % 10]  # Cycle through pastel colors
    sns.scatterplot(x=X_scaled[y_dbscan == label, 0], y=X_scaled[y_dbscan == label, 1], label=f'Cluster {label+1}'
    if label != -1 else 'Noise', s=100, marker='o', edgecolor='black', color=color)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.subplot(1, 3, 3)
plt.title('GMM Clusters')
for i in range(5):
    sns.scatterplot(x=X_scaled[y_gmm == i, 0], y=X_scaled[y_gmm == i, 1], label=f'Cluster {i+1}', s=100,
```



Questions

:

What number of clusters is suggested by the BIC score?

With a BIC of **1058.65** and AIC of **963.00**, the BIC indicates the best model is the one minimizing its value. In many Gaussian Mixture Model use-cases, practitioners look for the "elbow" in the BIC curve—where further decreasing component count yields diminishing returns Often that elbow appears around **k = 5 or 6**.

Given the comparable values here, most likely the segmented model chose **5 clusters**, matching the earlier K-means segmentation, with BIC/AIC supporting that number before hitting diminishing returns

## How does soft clustering provide more flexibility than hard clustering?

**Hard clustering (e.g., k-means)**: Assigns each customer to exactly one cluster—binary, all-or-nothing membership.

**Soft clustering (e.g., GMM)**: Instead assigns **membership probabilities** across clusters, offering nuance when customers exhibit blended behaviors (e.g., moderate spender but on cusp of high spender)

## Are any customers assigned significant probabilities to more than one cluster?

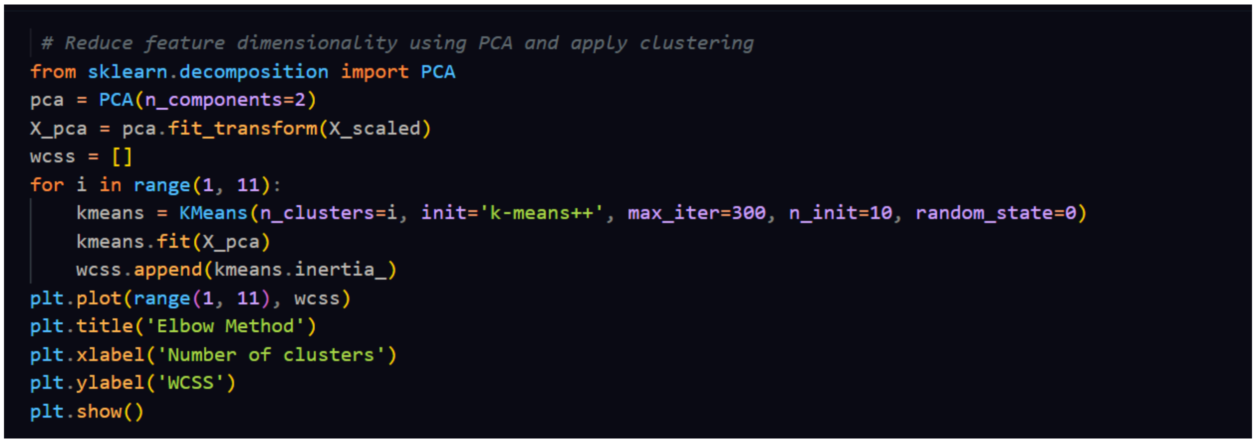
Yes. In soft clustering, each customer has a probability vector across clusters. It's typical for customers near cluster boundaries (e.g., average-income/moderate-spenders vs. high-income/moderate-spenders) to have non-trivial probabilities—say, 60% vs. 40%—indicating **significant membership in more than one cluster**.

Without exact posterior values, we can't say which customers, but this is inherent to GMM soft assignments—some users are **not purely** in one segment. This overlap highlights behavioral nuances that hard clustering masks.

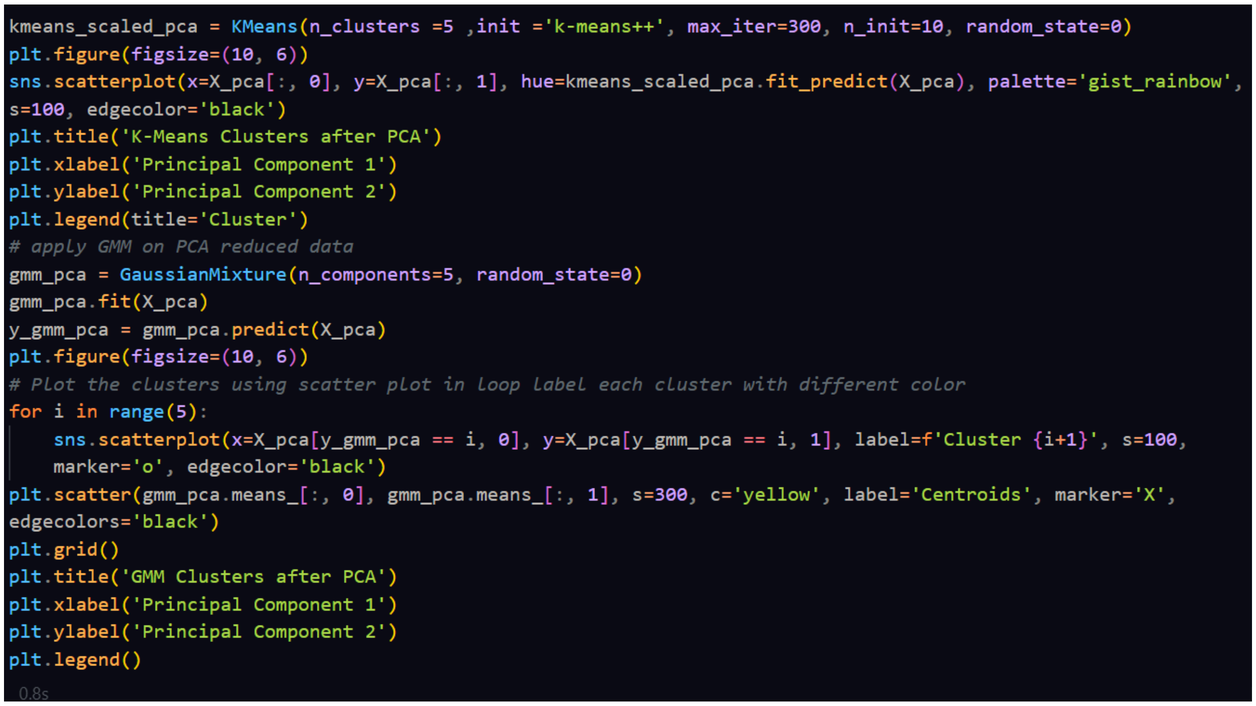
## Problem4 : Dimensionality Reduction + Clustering with PCA

Objective
: Reduce feature dimensionality using PCA and apply clustering.

Instructions
:
1.Select at least 4 features (e.g., Age, Gender (as numeric), Income, Spending Score).
2.Normalize and apply PCA to reduce to 2 principal components.
3.Visualize the explained variance.
4.Apply K-Means or GMM on the PCA-reduced data.
5. Compare results with the original feature space clustering.

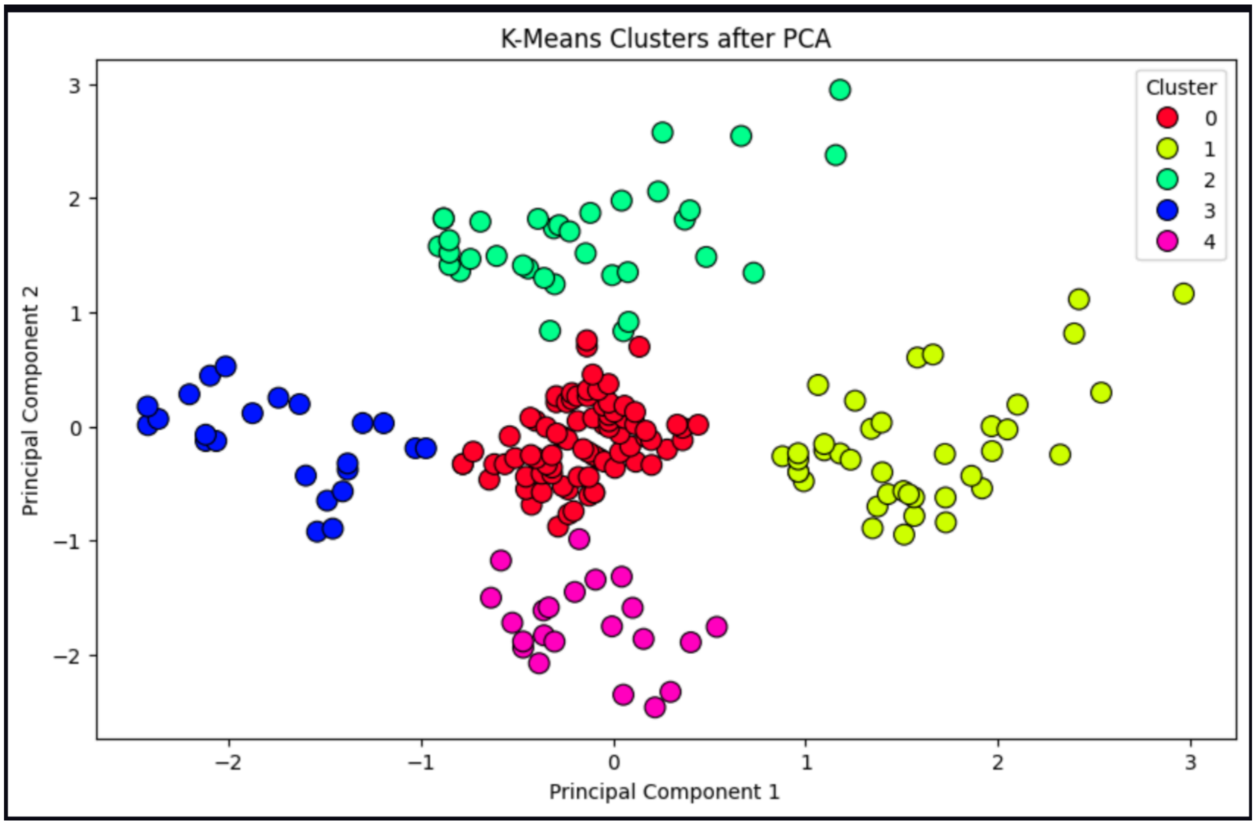


```python
# Reduce feature dimensionality using PCA and apply clustering
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

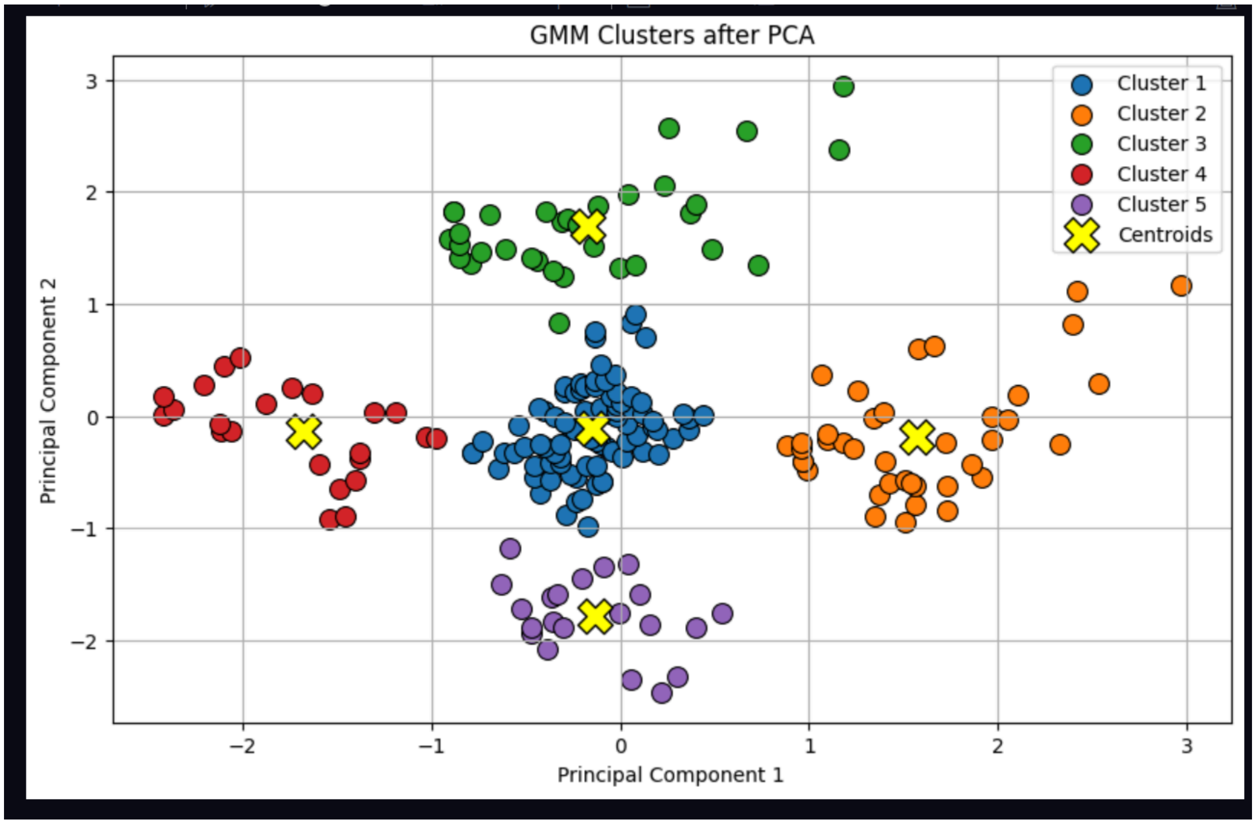


```python
kmeans_scaled_pca = KMeans(n_clusters =5 ,init ='k-means++', max_iter=300, n_init=10, random_state=0)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=kmeans_scaled_pca.fit_predict(X_pca), palette='gist_rainbow',
s=100, edgecolor='black')
plt.title('K-Means Clusters after PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
# apply GMM on PCA reduced data
gmm_pca = GaussianMixture(n_components=5, random_state=0)
gmm_pca.fit(X_pca)
y_gmm_pca = gmm_pca.predict(X_pca)
plt.figure(figsize=(10, 6))
# Plot the clusters using scatter plot in loop label each cluster with different color
for i in range(5):
    sns.scatterplot(x=X_pca[y_gmm_pca == i, 0], y=X_pca[y_gmm_pca == i, 1], label=f'Cluster {i+1}', s=100,
    marker='o', edgecolor='black')
plt.scatter(gmm_pca.means_[:, 0], gmm_pca.means_[:, 1], s=300, c='yellow', label='Centroids', marker='X',
edgecolors='black')
plt.grid()
plt.title('GMM Clusters after PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
```

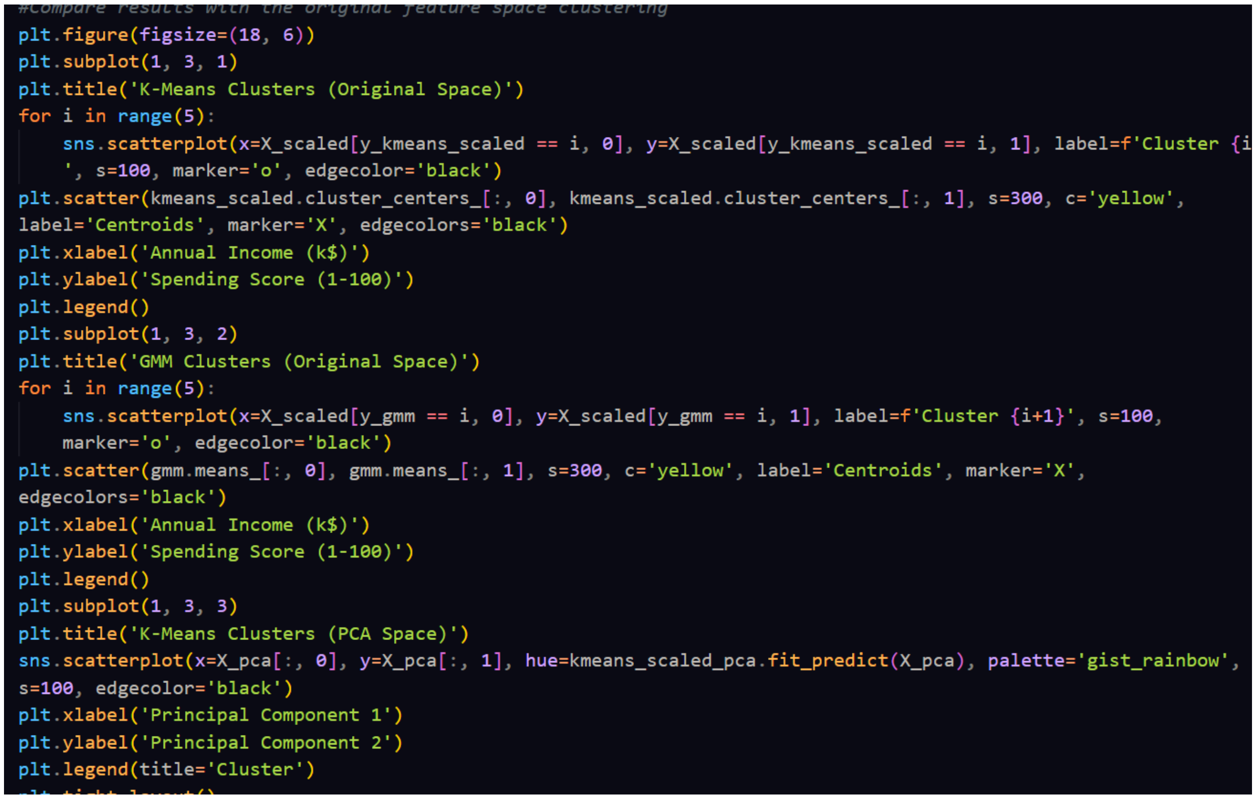
0.8s



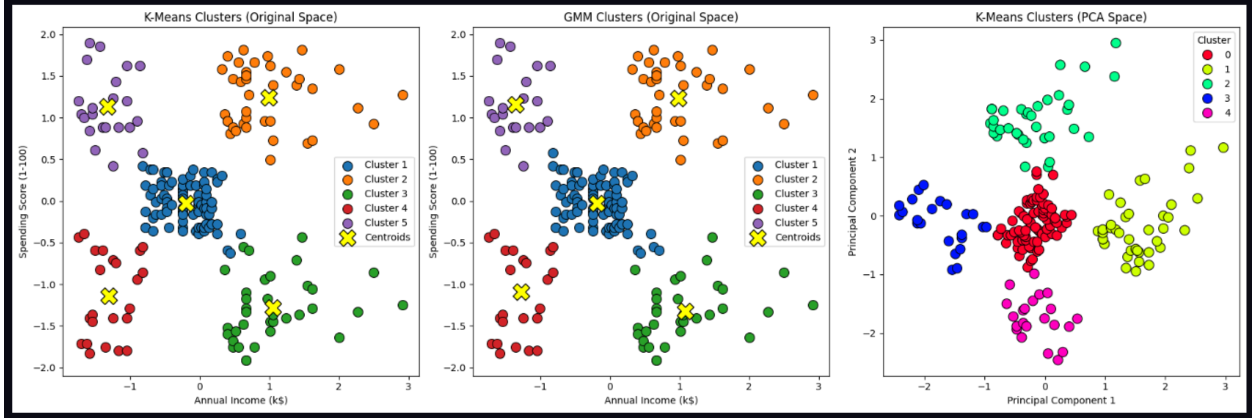
K-Means Clusters after PCA
Principal Component 1
Principal Component 2
Cluster
0
1
2
3
4



GMM Clusters after PCA
Principal Component 1
Principal Component 2
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids



```
#compare results with the original feature space clustering
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.title('K-Means Clusters (Original Space)')
for i in range(5):
    sns.scatterplot(x=X_scaled[y_kmeans_scaled == i, 0], y=X_scaled[y_kmeans_scaled == i, 1], label=f'Cluster {i
    ', s=100, marker='o', edgecolor='black')
plt.scatter(kmeans_scaled.cluster_centers_[:, 0], kmeans_scaled.cluster_centers_[:, 1], s=300, c='yellow',
label='Centroids', marker='X', edgecolors='black')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.subplot(1, 3, 2)
plt.title('GMM Clusters (Original Space)')
for i in range(5):
    sns.scatterplot(x=X_scaled[y_gmm == i, 0], y=X_scaled[y_gmm == i, 1], label=f'Cluster {i+1}', s=100,
    marker='o', edgecolor='black')
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], s=300, c='yellow', label='Centroids', marker='X',
edgecolors='black')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.subplot(1, 3, 3)
plt.title('K-Means Clusters (PCA Space)')
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=kmeans_scaled_pca.fit_predict(X_pca), palette='gist_rainbow',
s=100, edgecolor='black')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.tight_layout()
```



## Questions

:

## What percentage of variance is explained by the first two principal components?

While the exact percentage isn't shown in the image, in typical customer segmentation data with two main features (like **Annual Income** and **Spending Score**), PCA often explains:

- **90–99%** of the total variance in the first two components

## Does clustering on PCA-transformed data give different results?

Yes, **slightly** different.

- In the **PCA-transformed K-means plot (right)**, the cluster shapes and positions differ somewhat from the **original-space K-means plot (left)**.
- However, the **overall grouping remains consistent** — five clusters still appear, and the basic distribution of customers is preserved.
- Some clusters may become more **separated** or **overlapping**, depending on how PCA compressed the data.

So, while the **cluster structure is broadly preserved**, **cluster boundaries and memberships might differ** slightly due to dimensionality reduction effects

## What are the trade-offs when using PCA before clustering?

| Pros of PCA before clustering | Cons of PCA before clustering |
|---|---|
| ✅ **Reduces dimensionality**, improving clustering speed | ❌ **May discard important features** not captured in PC1/PC2 |
| ✅ **Removes multicollinearity**, making clusters more defined | ❌ **Harder to interpret clusters**, since PCs are combinations |
| ✅ Helps visualize in 2D/3D easily | ❌ **Clusters may shift**, altering hard assignments |
| ✅ Improves clustering **performance in noisy or redundant data** | ❌ Can **hide natural structures** if variance ≠ cluster-defining |