🤖 What is K-Means Clustering?

K-Means is an **unsupervised machine learning** algorithm used to find **clusters (groups)** in a dataset.

It is useful when we want to automatically group similar data points together **without having labeled data**.

🔧 How Does K-Means Work?

1. **Choose the number of clusters** (`K`) you want.
2. **Randomly select K points** as the initial **centroids** (cluster centers).
3. Assign each data point to the **nearest centroid**.
4. **Recalculate the centroids** by finding the average of all points assigned to each cluster.
5. **Repeat** steps 3–4 until the centroids stop changing (i.e., the algorithm converges).

📌 Example Use Cases:

- Grouping customers by purchasing behavior
- Organizing documents based on content similarity
- Identifying patterns in data with no labels

⚠️ Things to Know:

- You must choose the number of clusters `K` manually.
- K-Means works best with **numeric, continuous data**.
- It may give different results each time unless you fix the random seed.

In this notebook, we'll use K-Means on **simple synthetic 2D data** to see how it forms clusters.

Simple K-Means Project (Using Made-Up Data)

Simple K-Means Clustering Demo

This mini project demonstrates the K-Means clustering algorithm using **synthetic data** (artificially created for demo purposes). The data is 2-dimensional and visually easy to understand.

We'll:

- Generate random data points (blobs)
- Cluster them using K-Means
- Visualize the results

Libraries used:

- `sklearn` (for data generation and KMeans)

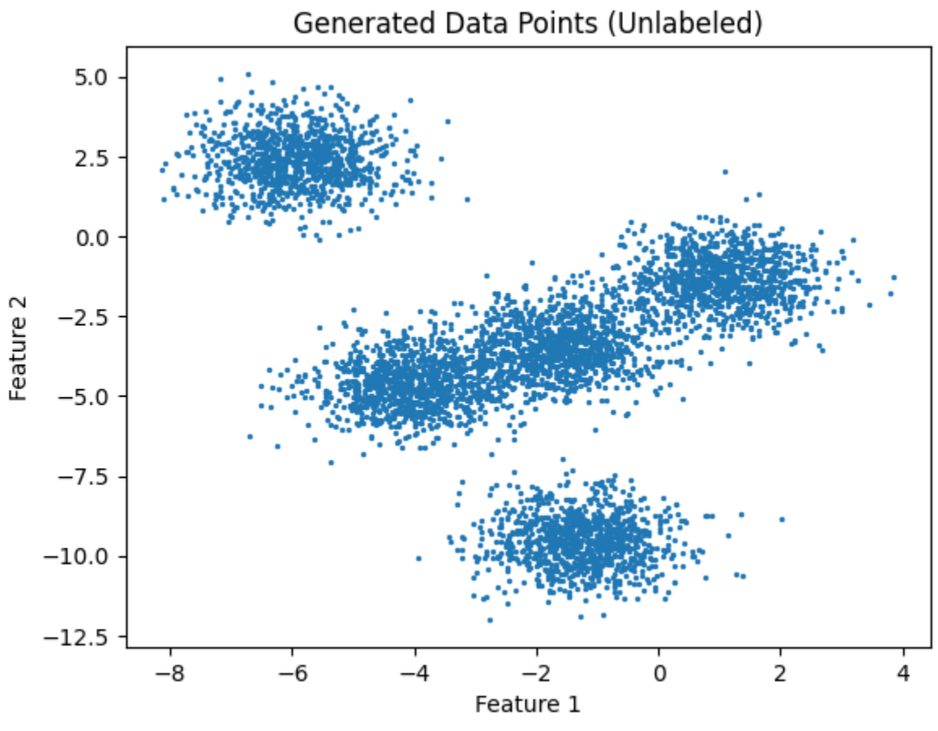
- `matplotlib` (for visualization)

Import Libraries

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

Generate Simple Dataset

In [100…
```
# Generate synthetic data (2D points)
X, y_true = make_blobs(n_samples=5000, centers=5, cluster_std=0.8, random_state=2)

# Visualize the data (no clustering yet)
plt.scatter(X[:, 0], X[:, 1], s=2)
plt.title("Generated Data Points (Unlabeled)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



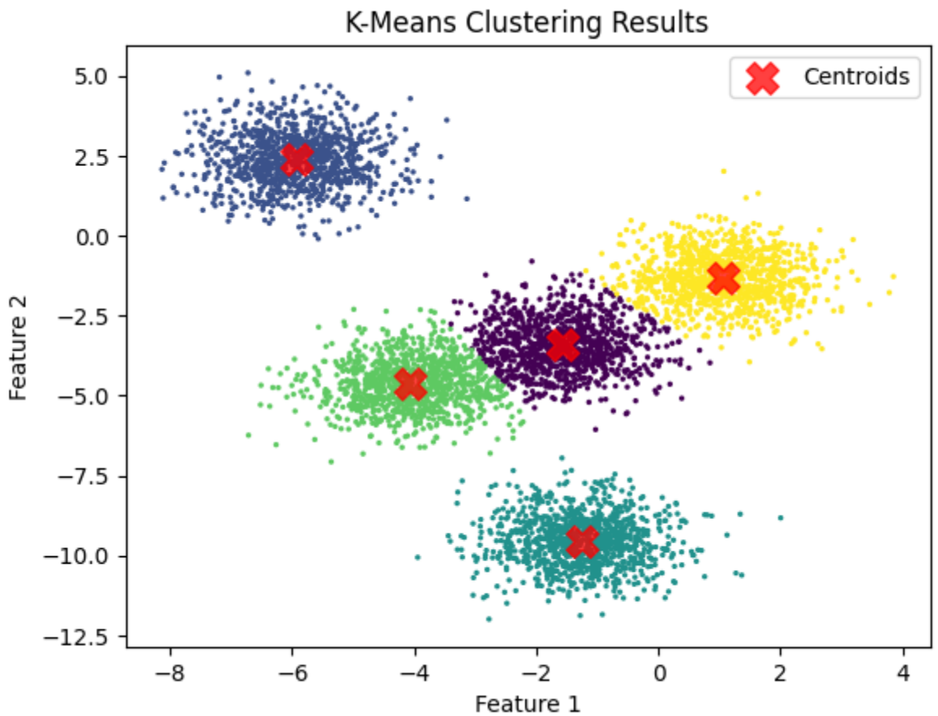
Apply K-Means Clustering

In [102…
```
# Apply KMeans with K=5
kmeans = KMeans(n_clusters=5, random_state=0)
kmeans.fit(X)
```

```
# Get cluster centers and labels
centers = kmeans.cluster_centers_
labels = kmeans.labels_
```

Visualize the Clusters

In [103…
```
# Plot clusters with their centroids
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=2)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X', label='Centroids')
plt.title("K-Means Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



Summary

- We created 5000 random data points grouped into 5 clusters.
- K-Means correctly identified the 5 natural clusters.
- The red X markers show the cluster centers (centroids).

This simple demo shows how K-Means can group data without any prior labels.

🛍️ Customer Segmentation using K-Means Clustering

📌 Project Overview

In this project, we perform **unsupervised machine learning** to segment customers of a mall based on their demographic and spending behavior. The aim is to group similar customers together so businesses can tailor their strategies (e.g., marketing, loyalty programs) to specific customer segments.

We use the **Mall Customers Dataset** which contains data on:

- **Gender**
- **Age**
- **Annual Income (k$)**
- **Spending Score (1-100)**

🧠 Machine Learning Techniques Used

- **Clustering Algorithm**: K-Means
- **Dimensionality Reduction**: Principal Component Analysis (PCA)
- **Model Evaluation**:
  - Elbow Method (for optimal number of clusters)
  - Silhouette Score (for cluster quality)

📂 Dataset Information

- **Name**: Mall Customers Dataset
- **Source**: Kaggle
- **Features**:
  - `CustomerID` : Unique ID
  - `Gender` : Male/Female
  - `Age` : Customer's age
  - `Annual Income (k$)` : Yearly income in thousands
  - `Spending Score (1-100)` : Score assigned by the mall based on spending behavior and income

---

🎯 Objective

To segment the customers into distinct groups using **K-Means Clustering**, and visualize the resulting clusters using **PCA**, enabling strategic business decisions.

📊 Expected Outcomes

- Identification of different customer segments.
- Visual representation of clusters in 2D space.
- Insights into customer types based on spending and income behavior.

✅ Let's begin with data loading and exploration!

Install & Import Required Libraries

In [3]:
```
# Install necessary libraries
# !pip install seaborn scikit-learn matplotlib

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

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
```

Load the Dataset

In [124…
```
# Load data from a CSV (hosted on GitHub Gist)
# Original source: https://www.kaggle.com/vjchoudhary7/customer-segmentation-tutorial

url = 'https://gist.githubusercontent.com/pravalliyaram/5c05f43d2351249927b8a3f3cc3e5ecf/raw/Mall_Customers.csv'
df = pd.read_csv(url)

# Select relevant features for clustering
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

# Display the first few rows
df.head()
```

Out[124…

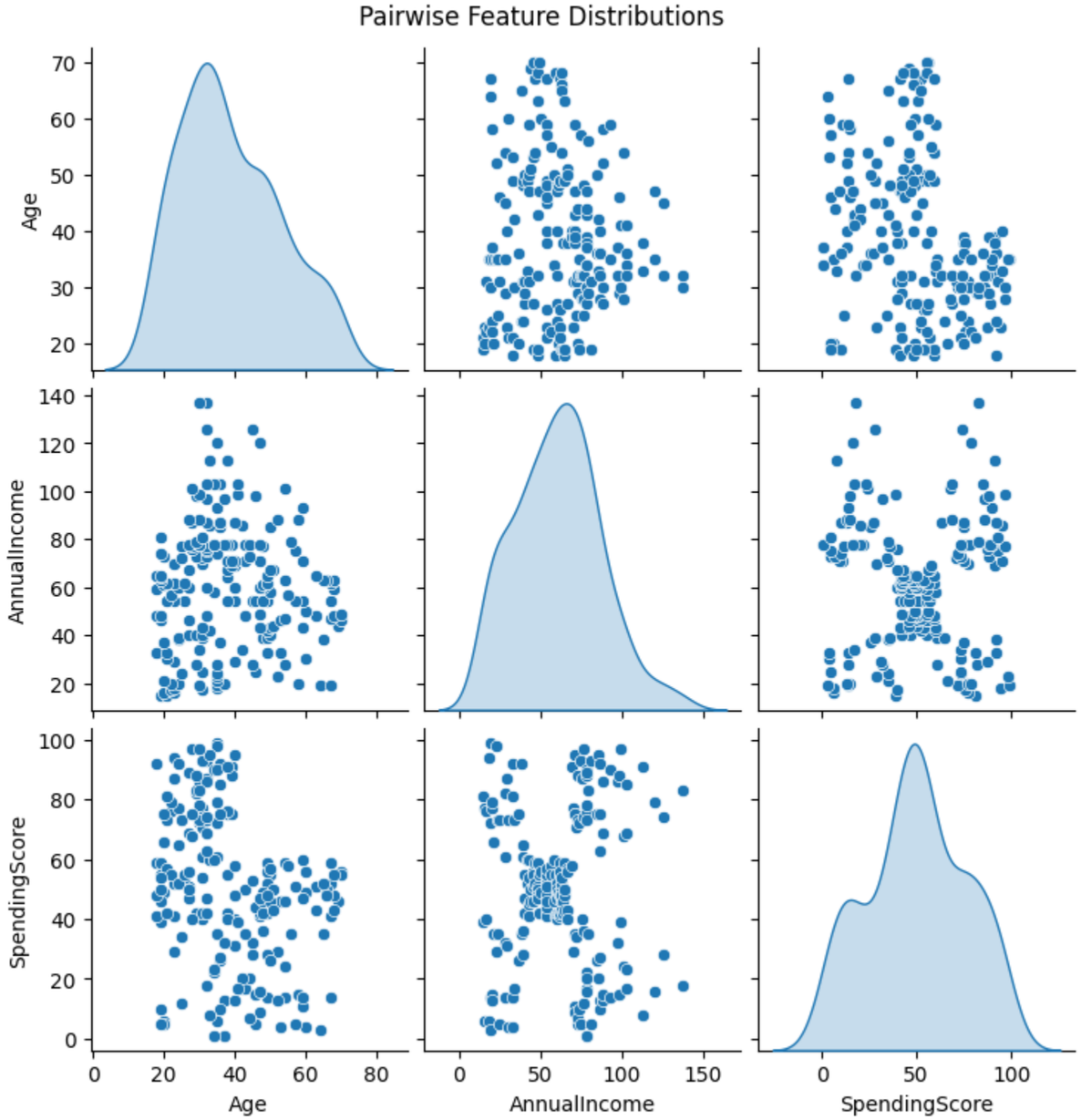
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

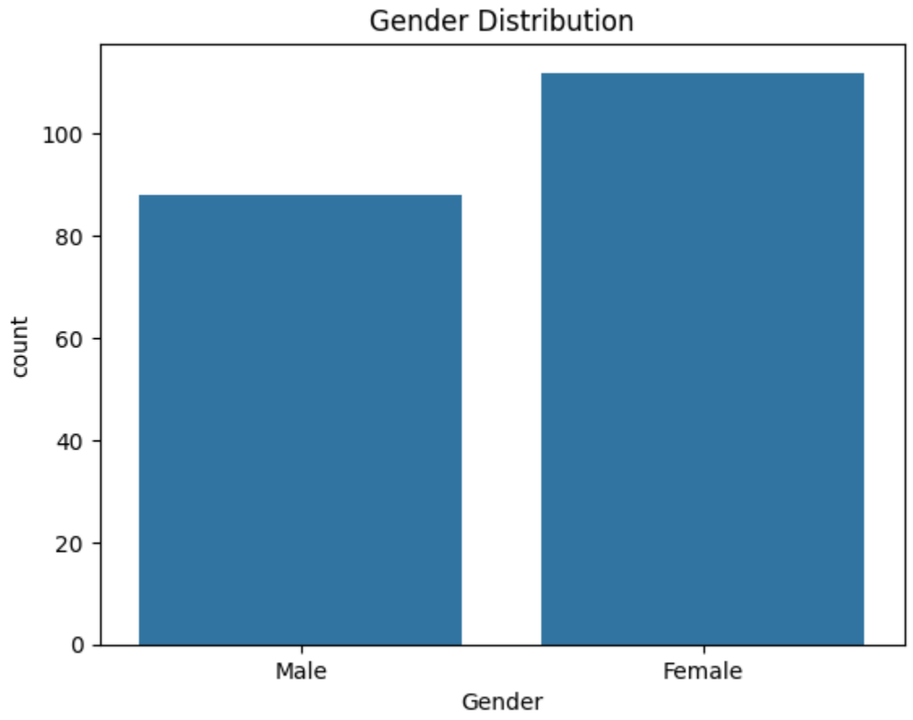
Exploratory Data Analysis (EDA)

In [125…
```
# Check dataset structure
df.info()
df.describe()

# Rename columns for convenience
df.columns = ['CustomerID', 'Gender', 'Age', 'AnnualIncome', 'SpendingScore']

# Select features for clustering after renaming
X = df[['AnnualIncome', 'SpendingScore']]

# Plot pairwise feature distributions
```

```
sns.pairplot(df[['Age', 'AnnualIncome', 'SpendingScore']], diag_kind='kde')
plt.suptitle("Pairwise Feature Distributions", y=1.02)
plt.show()

# Plot Gender distribution
sns.countplot(x='Gender', data=df)
plt.title("Gender Distribution")
plt.show()
```

```
<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
```


Pairwise Feature Distributions
Age
AnnualIncome
SpendingScore
Age
AnnualIncome
SpendingScore



Data Preprocessing

In [126…

```
# Convert categorical Gender column
df_encoded = df.copy()
df_encoded['Gender'] = df_encoded['Gender'].map({'Male': 0, 'Female': 1})

# Drop CustomerID
X = df_encoded.drop('CustomerID', axis=1)
X.head()
```

Out[126…

	Gender	Age	AnnualIncome	SpendingScore
0	0	19	15	39
1	0	21	15	81
2	1	20	16	6
3	1	23	16	77
4	1	31	17	40

Elbow Method to Determine Optimal K

What is the Elbow Method in K-Means?

The Elbow Method is a common technique used to determine the optimal number of clusters (K) when using the K-Means clustering algorithm.

Why is it important?

K-Means requires you to manually specify the number of clusters (K). Choosing too few clusters may group dissimilar points together, while too many clusters may result in overfitting. The Elbow Method helps identify a good balance.

How does it work?

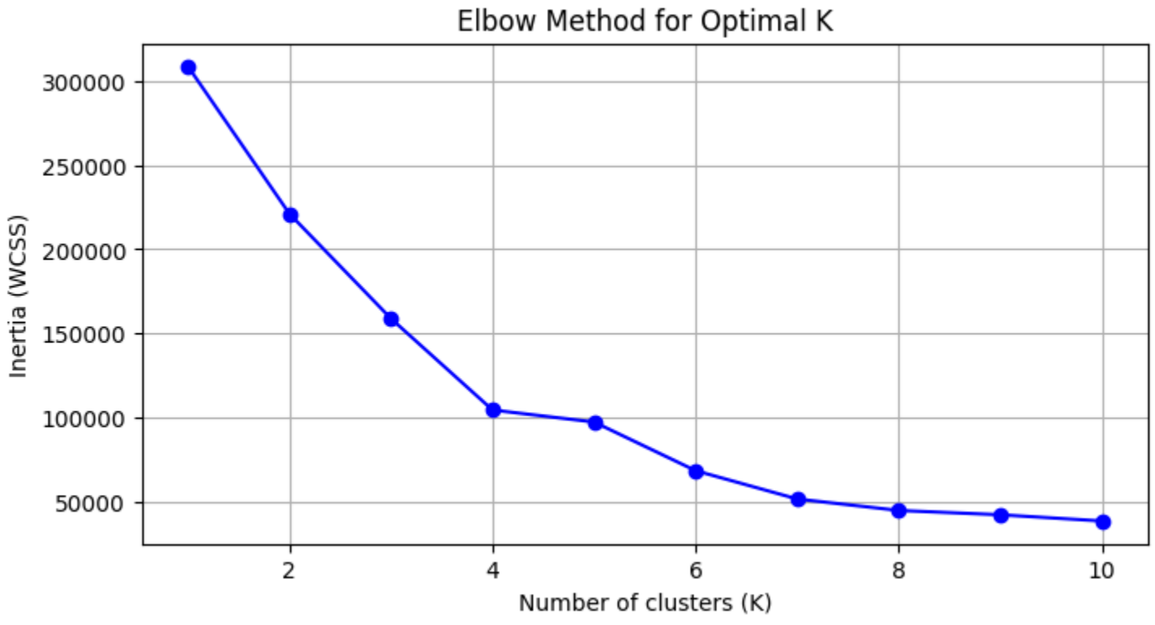
1. Run K-Means for a range of different values of K (e.g., from 1 to 10).
2. For each K, calculate the **inertia** (also known as **Within-Cluster Sum of Squares**, or WCSS), which measures how tightly the data points are grouped around the cluster centers.
3. Plot a graph of K (x-axis) vs. inertia (y-axis).
4. Look for the point where the decrease in inertia begins to slow down significantly. This point is called the "elbow."

Interpretation

The "elbow" point represents the optimal K — beyond this point, increasing the number of clusters provides little improvement in cluster compactness.

In [127…

```
inertia = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plotting the Elbow Curve
plt.figure(figsize=(8, 4))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Inertia (WCSS)')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```



Silhouette Score for Cluster Quality

What is the Silhouette Score in Clustering?

The Silhouette Score is a metric used to evaluate the quality of clusters created by a clustering algorithm such as K-Means.

Purpose

It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). A higher score indicates better-defined and more distinct clusters.

Score Range

- The score ranges from **-1 to 1**:
  - **+1**: The sample is far away from neighboring clusters and well-matched to its own cluster.
  - **0**: The sample is on or very close to the decision boundary between two clusters.
  - **-1**: The sample may have been assigned to the wrong cluster.

How It Works

For each data point:

- Compute the **average distance to all other points in the same cluster** (a).
- Compute the **average distance to all points in the nearest different cluster** (b).
- The Silhouette Score for the point is calculated as:
  (b - a) / max(a, b)

The overall Silhouette Score is the average of these scores across all points.

Interpretation

- A score closer to **1** indicates better clustering.
- A score near **0** means overlapping clusters.
- A score below **0** suggests that the points may be assigned to the wrong clusters.

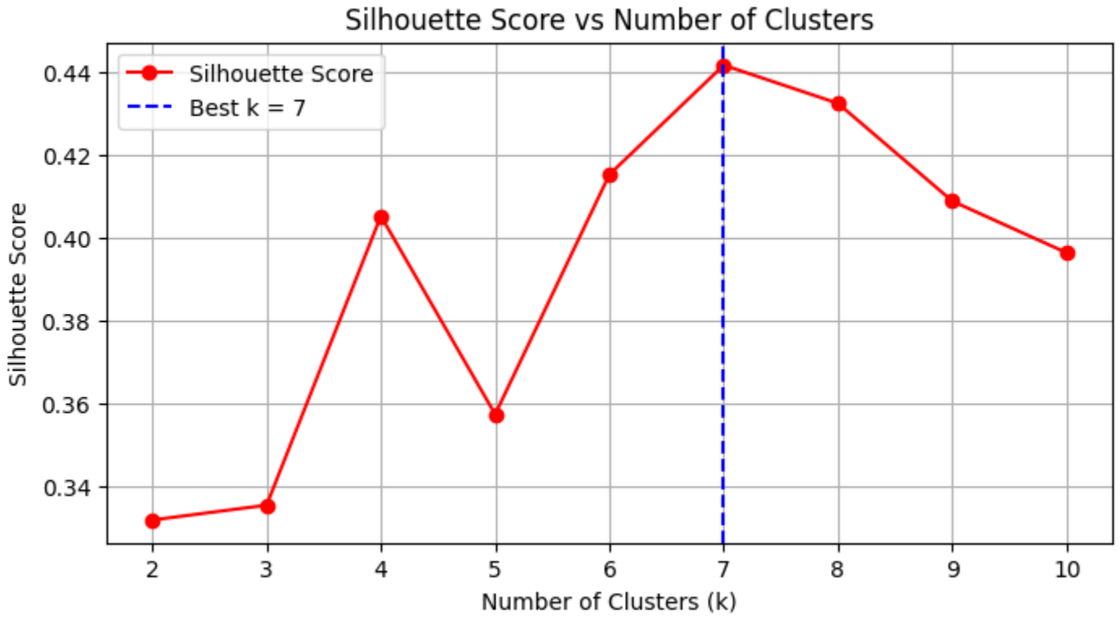
Summary

The Silhouette Score is a useful tool for validating the consistency and separation of clusters, especially when trying to determine the best number of clusters.

In [128…

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Step 1: Calculate silhouette scores for k = 2 to 10
silhouette_scores = []
K = range(2, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X)
    score = silhouette_score(X, labels)
    silhouette_scores.append(score)

# Step 2: Find the best k (with the highest silhouette score)
best_k = K[silhouette_scores.index(max(silhouette_scores))]
print(f"Best number of clusters based on Silhouette Score: {best_k}")

# Step 3: Plot silhouette scores
plt.figure(figsize=(8, 4))
plt.plot(K, silhouette_scores, 'ro-', label='Silhouette Score')
plt.axvline(x=best_k, color='blue', linestyle='--', label=f'Best k = {best_k}')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs Number of Clusters')
plt.legend()
plt.grid(True)
plt.show()
```

```
Best number of clusters based on Silhouette Score: 7
```



Final KMeans Model

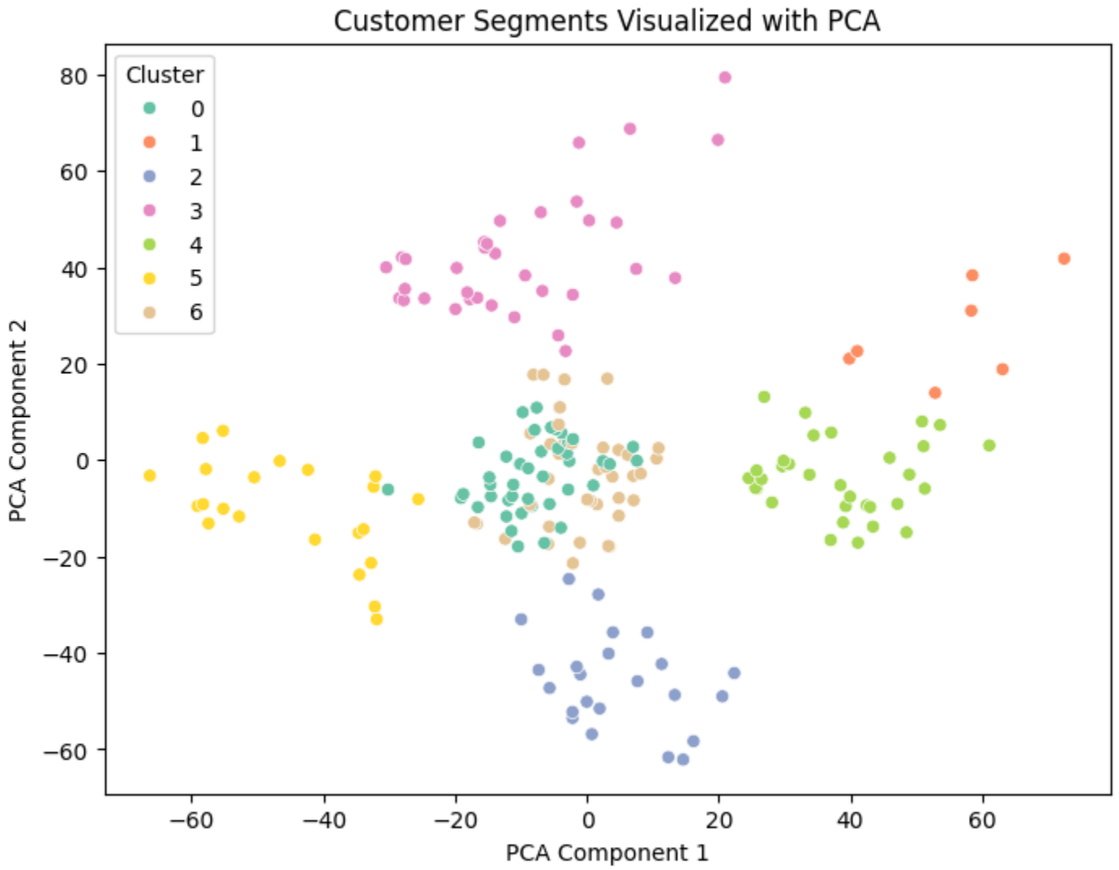
In [132…
```
# Use the best_k found from silhouette or elbow analysis
kmeans = KMeans(n_clusters=best_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)

df[['CustomerID', 'Cluster']].head()
```

Out[132…

	CustomerID	Cluster
0	1	5
1	2	2
2	3	5
3	4	2
4	5	5

PCA for 2D Visualization

In [133…
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=df['Cluster'], palette='Set2')
plt.title("Customer Segments Visualized with PCA")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```



Cluster Summary

In [134…

```
# Analyze mean feature values per cluster
df.groupby('Cluster')[['Age', 'AnnualIncome', 'SpendingScore']].mean().round(2)
```

Out[134…

Cluster	Age	AnnualIncome	SpendingScore
0	56.34	53.70	49.39
1	33.00	114.71	78.43
2	25.52	26.30	78.57
3	41.65	88.74	16.76
4	32.62	80.38	82.94
5	44.32	25.77	20.27
6	27.32	57.50	48.45